UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/838,653	04/19/2001	Christopher Jay Davia	RAL920000081US1	9232

7590 06/17/2004

ANDREW J. DILLON
BRACEWELL & PATTERSON, LLP
SUITE 350 LAKEWOOD ON THE PARK
7600B NORTH CAPITAL OF TEXAS HIGHWAY
AUSTIN, TX 78731

EXAMINER	
STEVENS, ROBERT	

ART UNIT	PAPER NUMBER
2176	

DATE MAILED: 06/17/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/838,653	DAVIA, CHRISTOPHER JAY	
	Examiner	Art Unit	
	Robert M Stevens	2176	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 April 2001*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-21* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☒ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-21* is/are rejected.

7)☒ Claim(s) *19-21* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 April 2001* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-21 are pending in Application No. 09/838,653, entitled "Method and Apparatus for the Separation of Web Layout, Logic, and Data when Used in Server-Side Scripting", filed April 19, 2001.

2. No IDS was filed as of the date of examination.

Drawings

3. The drawings are objected to because:

a. Figure 1 #126 is missing a reference pointer.

b. Figure 2 should be designated by a legend such as --Prior Art-- because only that which is old is illustrated. See MPEP § 608.02(g). Corrected drawing sheets are required in reply to the Office action to avoid abandonment of the application. The replacement sheet(s) should be labeled "Replacement Sheet" in the page header (as per 37 CFR 1.84(c)) so as not to obstruct any portion of the drawing figures. If the changes are not accepted by the examiner, the applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

c. Figures 2 and 3 contain descriptive/narrative language (See Fig 2: "Traditional Method ...", and Fig. 3: "The Server-Side Script ...", "The Engine Frame's ...", "The HTML and ..."), which must be removed from the drawings.

d. Figure 2 reference pointer 202 points to (impinges upon) "Next" box rather than "Web content" box.

e. Figure 2 elements "Img1" and "Img2" should have reference numbers (both in the figures and the specification, which refers to those elements).

f. Figure 2 reference pointer 206 points to (impinges upon) "Back" box rather than "next web page" box.

g. Figure 3 discussion of a "classification screen" element (page 11 line 28) is not reflected in the Figure itself.

4. A proposed drawing correction or corrected drawings are required in reply to the Office action to avoid abandonment of the application. The objection to the drawings will not be held in abeyance.

Specification

5. The disclosure is objected to because of the following informalities:

a. Please correct all grammatical/spelling/etc. errors throughout the specification, such as page 12 line 4 "and image" should be "an image".

Appropriate correction is required.

Claim Objections

6. **Claims 19-21 objected to** because of the following informalities:

System claim 19 refers to method claim 4. For examination purposes, the Office will consider claim 19 to depend upon claim 18.

System claim 20 refers to method claim 1. For examination purposes, the Office will consider claim 20 to depend upon claim 16.

System claim 21 refers to method claim 4. For examination purposes, the Office will consider claim 21 to depend upon claim 16.

7. Appropriate correction is required.

Claim Rejections - 35 USC § 112

8. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

9. Claims 3, 7, 10, 14, 17 and 21 are rejected under 35 U.S.C. 112, second paragraph, as failing to set forth the subject matter which applicant(s) regard as their invention.

Regarding claims 3, 10 and 17, the term "more efficient" is a relative term which renders each claim indefinite.

Regarding claims 7, 14 and 21, the terms "importing" and "exporting" were not defined in the specification, and appear to have the same meaning. As such, the scope of these claims are indefinite. For purposes of examination, the Office considers these terms to both mean "referencing".

Regarding claims 7, 14 and 21, the term "classifications" was not defined in the specification. As such, the scope of these claims are indefinite. For purposes of examination, the Office considers this term to mean " web page presentations".

Claim Rejections - 35 USC § 102

10. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

11. **Claims 1-3, 8-10, 16 and 17 are rejected under 35 U.S.C. 102(b)** as being anticipated by Douglis et al (US Patent No. 6,021,426, hereafter referred to as "Douglis").

Regarding independent (method) claim 1, Douglis discloses:

> *A method of providing content for display of a web page, comprising: generating a dynamic content frame separate from a static content frame comprising layout and logic information during development of said web page;* (col 3 line 66 thru col 4 line 34, including Table 1) *and downloading said dynamic content frame independently of said static content frame* (col 3 lines 36-44).

Regarding claim 2, which is dependent upon claim 1, Douglis discloses:

> *further comprising enabling a display of said web page by incorporating features of said dynamic content frame and said static content frame into a display frame* (col 4 lines 14-19).

Regarding claim 3, which is dependent upon claim 2, Douglis discloses:

wherein said downloading step restricts a download to only said dynamic content frame whenever said web page is requested to be downloaded in sequence with a second web page sharing similar layout and logic, whereby a previous static content frame is stored on the system requesting a download of said web page and more efficient download and display of said web page is achieved (col 3 lines 40-44).

Regarding independent (computer program product) claim 8: Claim 8 is substantially similar to method claim 1, and therefore likewise rejected.

Regarding claim 9, which is dependent upon claim 8: claim 9 is substantially similar to method claim 2, and therefore likewise rejected.

Regarding claim 10, which is dependent upon claim 9: claim 10 is substantially similar to method claim 3, and therefore likewise rejected.

Regarding independent (system) claim 16: Douglis discloses:

A system for providing content for display of a web page, comprising:
a server having a memory component with server software; (col 5 lines 49-58)
means for collecting a web browser to said server; (col 5 lines 42-44, describing an HTTP communications between client and server)
logic for generating a dynamic content frame utilizing a server-side scripting
software separate from a static content frame comprising layout and logic information
during development of a web page at said server; (col 3 line 66 thru col 4 line 34, including Table 1) *and*
logic for downloading said dynamic content frame independently of said static

content frame to said web browser via said collecting means (col 3 lines 36-44).

Regarding claim 17, which is dependent upon claim 16, Douglis discloses:

wherein said downloading step restricts a download to only said dynamic content frame whenever said web page is requested to be downloaded in sequence with a second web page sharing similar layout and logic, whereby a previous static content frame is stored on the system requesting a download of said web page and more efficient download and display of said web page is achieved (col 3 lines 40-44).

Claim Rejections - 35 USC § 103

12. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

13. Claims 6, 7, 13, 14, 20 and 21 are rejected under 35 U.S.C. 103(a) as being unpatentable over Douglis in view of Emily A. Vander Veer, JavaScript for Dummies, 3rd Edition, IDG Books Worldwide, ©2000, pages 219-248 (hereafter "Vander Veer").

Regarding claim 6, which is dependent upon claim 1, Douglis does not explicitly disclose the added limitation.

Vander Veer, however discloses:

further comprising initializing said static content frame with common functions utilized by all screens, wherein said common functions include how to display and hide an image within a layer (pp. 244-246, especially code listings 12-3 and 12-4 and code output in Figures 12-6 and 12-7), how to write text into said layer (pp. 229-231, especially code listing 11-3 and output in Figure 11-7 showing the addition of the text "Computer"), and how to move said layer (pp. 240-243, code listing 12-2 and Figures 12-3 through 12-5).

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Vander Veer for the benefit of Douglis because to do so would allow a user to click a button and see a different image, or slide, without popping to another Web page as taught by Vander Veer at the first paragraph under "Tickets to a Slide show" on page 240.

Regarding claim 7, which is dependent upon claim 6, Douglis does not explicitly disclose the added limitation.

Vander Veer, however discloses:

further comprising: importing a re-usable Javascript function library from a generated HTML to support a unique behavior of said display frame (pp. 227-228 section entitled "Checkin' out the library, especially the code listings on page 227 and their description in the first paragraph on page 228); *and*

caching said Javascript function library in a web browser with which said web page is downloaded, wherein said library may be utilized for subsequent classifications (p. 228, especially second bullet "Improves performance time").

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Vander Veer for the benefit of Douglis because to do so would allow for a single copy of a JavaScript library to be loaded into memory, where the JavaScript interpreter could access it, no matter how many web pages reference

that library (thus improving performance time) as taught by Vander Veer at the second bullet labeled "Improves performance time" on page 228.

Regarding claims 13 and 20, these claims are substantially similar to claim 6 and thus are similarly rejected.

Regarding claims 14 and 21, these claims are substantially similar to claim 6 and thus are similarly rejected.

14. Claims 4, 5, 11, 12, 18 and 19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Douglis in view of Vander Veer as applied to claims 1, 8 and 16 above, and further in view of Laura Lemay, SAM's Teach Yourself Web Publishing with HTML 4 in 21 Days, 2nd Edition, Sam's Publishing, ©2000, XXX pp. 471-489 (hereafter "Lemay").

Regarding claim 4, which is dependent upon claim 1, Douglis discloses:

> wherein said generating step comprises generating an HTML page with two frames, wherein:
> a dynamic content frame created utilizing server- side scripting language that maintains changing information (col 2 lines 20-28)

However, Douglis does not explicitly recite the rest of that limitation.

Lemay, though, does teach the use of different data types:

including text data, images, and audio data (pp. 400-401, especially the code listing of page 400 incorporating text ["An oral family history ... tornado of 1903"], images [img src="soundicon.gif"], and audio data [a href="tornado.wav"].

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Lemay for the benefit of Douglis because to do so enabled one to assemble a media archive, which provides quick access to images or other media files for viewing and downloading as taught by Lemay on page 395 (see first paragraph under "Exercise 13.1: Creating a Family history Media Archive").

Douglas does not explicitly disclose the next limitation.

However, Vander Veer discloses:

> *a static content frame maintains layout information of a web page content and*
> *references an HTML file that includes a Javascript library to add, move, remove and*
> *change text and images in a display layer of said web page* (pp. 229-231, especially listing 11-3 and figures 11-6 and 11-7 for text manipulations, and pp. 240-246, especially listings 12-2 and 12-3 and figures 12-3 through 12-7 for image manipulations).

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Vander Veer for the benefit of Douglis because to do so would allow for a single copy of a JavaScript library to be loaded into memory, where the JavaScript interpreter could access it, no matter how many web pages reference that library (thus improving performance time) as taught by Vander Veer at the second bullet labeled "Improves performance time" on page 228.

Regarding claim 5, which is dependent upon claim 4, Douglis discloses:

> further comprising merging features of said static content frame and said dynamic content frame utilizing linking information provided within said dynamic content frame (col 6 lines 22-24),

However, Douglis does not explicitly disclose the next limitation.

Vander Veer, though, discloses:

> wherein said linking information comprises HTML/Javascript function calls to said static content frame (pp. 227-228, especially the code listings on page 227 and the descriptive paragraphs immediately before (p. 227) and after (p. 228) the code listings).

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Vander Veer for the benefit of Douglis because to do so would allow a user to click a button and see a different image, or slide, without popping to another Web page as taught by Vander Veer at the first paragraph under "Tickets to a Slide show" on page 240.

Regarding claims 11 and 18, these claims are substantially similar to claim 4 and thus are similarly rejected.

Regarding claims 12 and 19, these claims are substantially similar to claim 5 and thus are similarly rejected.

15. Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Lemay (pp. 471-489) in view of Cohen (US Patent No. 6,263,352, filed Nov 14, 1997).

Regarding independent (method) claim 15, Lemay discloses:

> *A method for extending interaction between static and dynamic content of a web page comprising:*
> *creating said web page with individual layers corresponding each to web page*
> *content* (p. 481, especially the <meta> tag attribute "content"), *and web page layout* (p. 480, especially the bottom third of the page discussing setting a color attribute) *and logic* (p. 478, especially the bottom fourth of the page discussion conditional branching and logical operators);

However, Lemay does not explicitly disclose the next limitation.

Cohen, though, does disclose:

> *and enabling manipulation of each layer by server- scripting software, wherein*
> *web page content may be changed by standard off-the-shelf scripting applications,*
> *without consideration of HTML and DHTML* (col 3 lines 60-64, discussing the use ASP,
> Microsoft's standard off-the-shelf server-side scripting application. Note, as addressed in the 112 section, the Office is unsure of the meaning of "without consideration of HTML and DHTML", and has ignored this phrase for examination purposes since such phrase may actually be adding a limitation which renders the claim ineffective).

It would have been obvious to one of ordinary skill in the art at the time of the invention to apply the teachings of Cohen for the benefit of Lemay because employing ASP "allows a user to write Web pages using a combination of a hypertext language ... and a scripting language" as taught by Cohen at col 2 lines 32-36.

Conclusion

16. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Non-patent Literature

Bradbury, Danny, "Scripting Languages", ComputerWeekly.com, Nov. 16, 2000 (www.computerweekly.com/Article21635.htm).

Burridge, David, "Browser targeted Cascading Style Sheets using JavaScript", The Complete Webmaster, Jan. 28, 1999 (www.abiglime.com/webmaster/articles/jscript/012899.htm).

Halasz, Steven, "An Improved Method for Creating Dynamic Web Forms Using APL", APL Berlin 200 Proceedings, pp. 104-111, APL00, 07/00, Berlin, Germany, [ACM 1-58113-182-8/01/0007].

Labrinidis et al., "Generating dynamic content at data-backed web servers: cgi-bin vs mod_perl", SIGMOD Record, vol 29, no. 1, March 2000, pp. 26-31.

Microsoft Corporation, "Dynamic HTML: The Next Generation of User Interface Design Using HTML", Feb. 1, 1997 (www.microsoft.com/technet/itsolutions/intranet/build/dhtml01.mspx).

Savio, Nadav, "What is Dynamic HTML?", Oct. 2, 1997 (hotwired.lycos.com/webmonkey/geektalk/97/39/index3a.html?tw=authoring.htm).

Stanek, William R., "Explore Creativity and Control With Dynamic HTML", adapted from PC Magazine, Jan. 20, 1998 (www.computersmiths.com/educat/dhtml/CreateandControl.htm).

US Patent Application Publications

Hayko et al	US 2002/0095522

US Patents

Bookman et al	5,761,673
Hill et al	6,023,714
Chadha et al	6,061,698
Cragun et al	6,161,112
Hagenaars	6,182,093
Guthrie	6,266,681
Kraus et al	6,266,684
Levine et al	6,405,221

17. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert M Stevens whose telephone number is (703) 605-4367. The examiner can normally be reached on M-F 7:30 - 4:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Joseph Feild can be reached on (703) 305-9792. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Robert M. Stevens
Art Unit 2176
Date: June 8, 2004

JOSEPH FEILD
SUPERVISORY PATENT EXAMINER

rms

Notice of References Cited	Application/Control No. 09/838,653	Applicant(s)/Patent Under Reexamination DAVIA, CHRISTOPHER JAY	
	Examiner Robert M Stevens	Art Unit 2176	Page 1 of 3

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-2002/0095522	07-2002	Hayko et al.	709/311
	B	US-5,761,673	06-1998	Bookman et al.	719/311
	C	US-6,021,426	02-2000	Douglis et al.	709/200
	D	US-6,023,714	02-2000	Hill et al.	715/513
	E	US-6,061,698	05-2000	Chadha et al.	715/513
	F	US-6,161,112	12-2000	Cragun et al.	715/501.1
	G	US-6,182,093	01-2001	Hagenaars, Harald H. J.	715/513
	H	US-6,263,352	07-2001	Cohen, Michael A.	715/513
	I	US-6,266,681	07-2001	Guthrie, John	715/501.1
	J	US-6,266,684	07-2001	Kraus et al.	715/513
	K	US-6,405,221	06-2002	Levine et al.	715/501.1
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Bradbury, Danny, "Scripting Languages", ComputerWeekly.com, Nov. 16, 2000 (www.computerweekly.com/Article21635.htm).
	V	Burridge, David, "Browser targeted Cascading Style Sheets using JavaScript", The Complete Webmaster, Jan. 28, 1999 (www.abiglime.com/webmaster/articles/jscript/012899.htm).
	W	Halasz, Steven, "An Improved Method for Creating Dynamic Web Forms Using APL", APL Berlin 200 Proceedings, pp. 104-111, APL00, 07/00, Berlin, Germany, [ACM 1-58113-182-8/01/0007].
	X	Labrinidis et al., "Generating dynamic content at data-backed web servers: cgi-bin vs mod_perl", SIGMOD Record, vol 29, no. 1, March 2000, pp. 26-31.

*A copy of this reference is not being furnished with this Office action (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Notice of References Cited	Application/Control No. 09/838,653	Applicant(s)/Patent Under Reexamination DAVIA, CHRISTOPHER JAY	
	Examiner Robert M Stevens	Art Unit 2176	Page 2 of 3

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-			
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Lemay, Laura, SAMs Teach Yourself Web Publishing with HTML 4 in 21 Days, 2nd Edition, Sams Publishing, © 2000, pp. 392-412 and 471-489.
	V	Microsoft Corporation, "Dynamic HTML: The Next Generation of User Interface Design Using HTML", Feb. 1, 1997 (www.microsoft.com/technet/itsolutions/intranet/build/dhtml01.mspx).
	W	Savio, Nadav, "What is Dynamic HTML?", Oct. 2, 1997 (hotwired.lycos.com/webmonkey/geektalk/97/39/index3a.html?tw=authoring.htm).
	X	Stanek, William R., "Explore Creativity and Control With Dynamic HTML", adapted from PC Magazine, Jan. 20, 1998 (www.computersmiths.com/educat/dhtml/CreateandControl.htm).

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Notice of References Cited	Application/Control No. 09/838,653	Applicant(s)/Patent Under Reexamination DAVIA, CHRISTOPHER JAY	
	Examiner Robert M Stevens	Art Unit 2176	Page 3 of 3

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-			
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Vander Veer, Emily A., JavaScript for Dummies, 3rd Edition, IDG Books Worldwide, © 2000, pp. 219-231 and 235-248.
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Laura Lemay
With revisions by Denise Tyler

SAMS
Teach Yourself

Web Publishing with HTML 4 in 21 Days

SECOND EDITION

SAMS
A Division of Macmillan USA
201 West 103rd St., Indianapolis, Indiana, 46290 USA

Sams Teach Yourself Web Publishing with HTML 4 in 21 Days, Second Edition

International Standard Book Number: 0-672-31725-7

Library of Congress Catalog Card Number: 99-63541

Printed in the United States of America

First Printing: December 1999

01 00 4 3

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability or responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Acquisitions Editor
Jeff Schultz

Development Editor
Damon Jordon

Managing Editor
Charlotte Clapp

Project Editor
George E. Nedeff

Copy Editor
Jill Bond

Indexer
Christine Nelsen

Proofreader
Maryann Steinhart

Technical Editor
Will Kelly

Interior Designer
Gary Adair

Cover Designer
Aren Howell

Team Coordinator
Amy Patton

Copy Writer
Eric Borgert

Production
Dan Harris
Mark Walchle

PART 5

DAY 13

Multimedia: Adding Sounds, Videos, and More

Learning how to integrate multimedia into your Web pages is as simple as creating hyperlinks to sound or video files. Presto! You have added multimedia to your Web site. That's not the whole story, of course. Aside from linking to multimedia files, you also can embed them in your Web pages. Unfortunately, embedding them can be a little tricky. While you only need to learn a few HTML elements, the multimedia-related HTML elements suffer from what seems like schizophrenia. They either are implemented differently in Microsoft Internet Explorer and Netscape Navigator, not supported at all in one or the other of the two browsers, or are a part of the HTML standard to which no one seems to be paying attention. In addition, there are quite a few competing audio and video formats available today. It's almost impossible to learn the ins and outs of each one before more appear with the promise of being the "be all and end all" of multimedia.

Even with recent advances in communications speed (the current average is 56Kbps for most home modems), improved sound and video compression/decompression technologies (MP3 audio files come to mind), and

powerful audio and video adapter cards, the Web is not the sound and video showcase that multimedia proponents dream of—not yet anyway.

Part of the problem of coming to grips with this fact is the incongruity between what we know today's computers are capable of and what we think the Web should deliver. Pop a CD or DVD disk in your drive and, blammo, 3D graphics, stereo surround-sound, and full-screen 30-frames-per-second digital video jump out and assault your auditory and visual senses without letting up until you slump over in a blathering heap of multimedia overload. Contrast that with most multimedia on the Web and you can be sorely disappointed. Low-quality sound, small video sizes, and long download times are par for the course.

Things are getting better. Witness the advent of MP3 audio file and Macromedia Flash animations and their increasing popularity. Each offers a low-bandwidth high-quality multimedia option; however, there is a price to pay for the progress being made. Namely, as Web users, we are being deluged with audio and video formats each requiring special plug-ins or helper applications. As a Web developer, you have to purchase expensive audio/video equipment and software in order to create your own multimedia content.

Having said all this, I will try to strike a balance in this lesson between showing you the techniques you can immediately use and the technologies that require you to devote a significant amount of time and energy in order apply. You will learn to accomplish the following:

- Create links to audio and video files so that visitors can download or play them
- Use the `embed` and `object` elements to include sound and video files in Web pages
- Learn how to embed QuickTime, Shockwave, Flash, and RealAudio or RealVideo files into your Web pages
- Use some of the unique multimedia capabilities of Microsoft Internet Explorer
- Recognize the most popular multimedia file types and the plug-ins or helper applications they require

Understanding How to Present Sound and Video

Despite all the complexity surrounding multimedia files and the number of formats available, it all boils down to choosing a method to integrate them into your Web pages. You can choose to create hyperlinks to those files or embed them directly into your Web pages. Linking to files is relatively foolproof but embedding them can be problematic.

A linked sound or video file, no matter what type of sound or video file it is, has a hyperlink to the source file within the Web page. When you click on a linked sound or video file, one of three possible events occur. First, you can download the file and save it to your computer. This method enables you to listen or view at a later time your file in whatever application you choose. Secondly, the file can download and automatically launch a helper application or plug-in to play. This occurs when the file is a recognized type, and a suitable player or plug-in is configured to play the file. Thirdly, if the file is recognized as streaming audio or video, a player is launched as a separate process that will begin to play the file as it downloads.

Embedded sound and video are integrated into the Web browser itself and are played with the help of plug-ins or helper applications. Embedded sound and video offer fewer options for the user because the file is integrated into the Web page itself. Most often, a helper-application or plug-in interface is created in the Web browser's window to play the file.

Without much further ado, let's get your elbows dirty and add some sound and video to Web pages.

The Old Standby: Linking

The sure-fire way to include multimedia files in your Web pages is to provide a hyperlink to them, which is supported by all versions of all browsers. People can decide whether they want to download the file and either listen to or view it at their convenience.

A common technique is to link to the file and provide a thumbnail preview of the media clip, a description, and the file size. This is considered a common courtesy so that people can estimate the download time. You also should provide links to any players required so that people can download the appropriate player, should they need it.

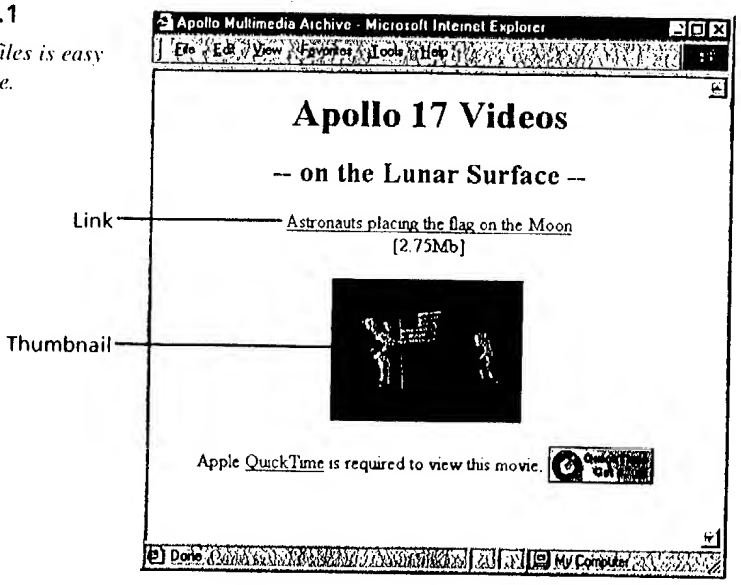
If I have a QuickTime video that I want to share, for example, I might fashion the code as the following:

```
<div align="center">
<h1>Apollo 17 Videos</h1>
<p><a href="Apollo_17_Flag.qt">Astronauts placing the flag on the Moon</a><br>
[2.75Mb]</p>
<img src="Apollo_17_Flag.gif" align="texttop" width="160" height="120" />
<p>Apple <a href="http://www.apple.com/quicktime">QuickTime</a> is required
to view this movie.
<a href="http://www.apple.com/quicktime"><img src="getquicktime4.gif" border="0"
align="absmiddle" width="88" height"31" /></a></p>
</div>
```

Figure 13.1 shows the resulting Web page.

FIGURE 13.1

Linking to files is easy and effective.



It also is considered good form to provide multiple types of media to download, should your visitors have a preference.

```
<html>
<head><title>Apollo Multimedia Archive</title></head>
<body>
<div align="center">
<h1>Apollo 17 Videos</h1>
<p>Astronauts placing the flag on the Moon</p>
<table border="0">
    <tr>
        <td rowspan="3"><img src="Apollo_17_Flag.gif" width="160" height="120"
/></td>
        <td><a href="Apollo_17_Flag.qt">QuickTime</a> [2.75Mb]</td>
    </tr>
    <tr>
        <td><a href="Apollo_17_Flag.mpg">MPEG</a> [2.45Mb]</td>
    </tr>
    <tr>
        <td><a href="Apollo_17_Flag.avi">AVI</a> [3.11Mb]</td>
    </tr>
</table>
<br />
<a href="http://www.apple.com/quicktime">
<img src="getquicktime4.gif" width="88" height"31" border="0"
```

```
alt="Get QuickTime" vspace="7" /></a>
<br>
<a href="http://microsoft.com/windows/mediaplayer/download/default.asp">
<img src="getmedia_white.gif" width="65" height="57" border="0"
alt="Get Windows Media Player" vspace="7" /></a>
<br />
</div>
</body>
</html>
```

Figure 13.2 shows the resulting Web page.

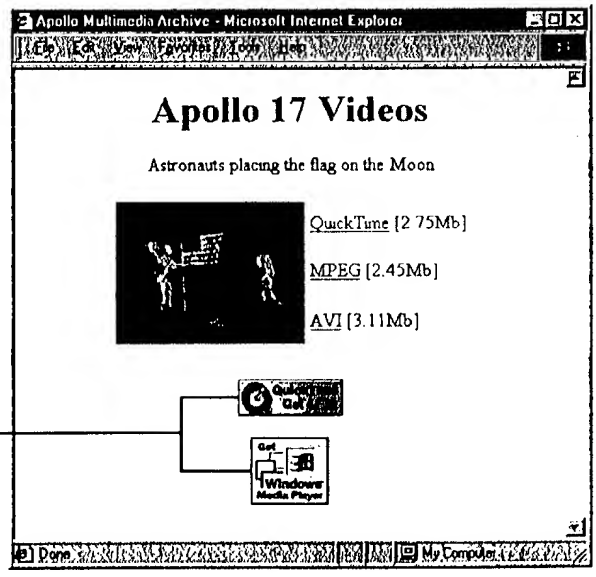


FIGURE 13.2
When linking sound and video, provide multiple formats if possible.

Exercise 13.1: Creating a Family History Media Archive

One of the common types of pages available on the Web is a *media archive*. A media archive is a Web page that serves no purpose other than to provide quick access to images or other media files for viewing and downloading.

Before the Web became popular, media such as images, sounds, and video were stored in FTP or Gopher archives. The text-only nature of these sorts of archives makes it difficult for people to find what they're looking for, as the filename usually is the only description they have of the content of the file. Even reasonably descriptive filenames, such as `the-trees-last-fall.gif` or `ave-maria.wav`, aren't very useful when you're talking about images or sounds. Although they may describe the files, the only way people actually can sample them is to go through the process of downloading the entire file and playing it.

- By using inline images as thumbnails and splitting up sound and video files into small "sample" clips with larger files, you can create a media archive on the Web that is far more usable than any of the text-only archives.

Note

Keep in mind that this sort of archive, with its heavy use of inline graphics and large media files, is optimally useful in graphical browsers attached to fast networks.

In this exercise, you'll create a simple example of a media archive with several GIF and JPEG images, WAV sounds, and a mixture of MPEG and AVI video.

By using your favorite image editor, you can create thumbnails of each of your pictures to serve as the inline icons and then insert `<img>` links in the appropriate spots in your archive file.

First, start with the framework for the archive, and then add a table for the thumbnail images, as in the following:

INPUT

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">

<html>
<head>
<title>My Family History</title>
</head>
<body>
<h1>My Family Media Archive</h1>

<div align="center">
<table border="0">
<tr>
    <td width="80"><h2>Images</h2></td>
    <td><p>Select an image to view it in a larger size</p></td>
</tr>
<tr>
    <td width="80">A bunch of family members in the early 1950s.</td>
    <td><img src="groupoldsmall.gif" height="103" width="150" alt="An old
group photo" /></td>
</tr>
<tr>

    <td width="80">Aunts Betsy and Phyllis sitting on the porch.</td>
    <td><img src="auntssmall.gif" height="96" width="150" alt="Two aunts
on a porch" /></td>
</tr>
```

```
<tr>
    <td width="80">Don when he was a child.</td>
    <td><img src="donoldsmall.gif" height="100" width="61" alt="Young
Don" /></td>
</tr>
</table>
</div>

</body>
</html>
```

Note that I include values for the `alt` attribute to the `<img>` tag, which will be substituted for the images in browsers that cannot view these images. Although you may not intend for your Web page to be seen by non-graphical browsers, at least offering a clue to people who stumble onto it is polite. This way, everyone can access the media files you're offering on this page.

Figure 13.3 shows how the page looks so far.

OUTPUT

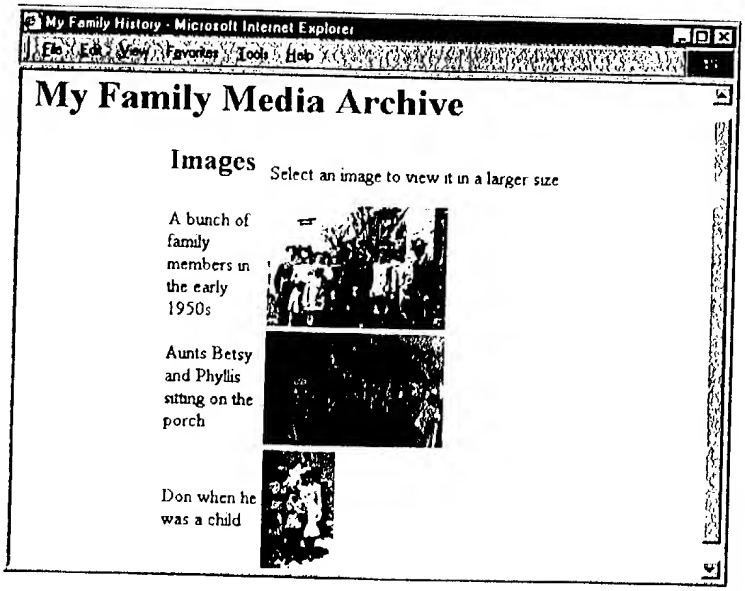


FIGURE 13.3
The Web page with the image archive almost completed.

The next step is to create the hyperlinks to each image that point to the larger file. By clicking them, people can choose either to view these files or download them to their computers.

INPUT

```
...
<tr>
    <td width="80">A bunch of family members in the early 1950s.</td>
    <td><a href="groupoldlarge.jpg">
    <img src="groupoldsmall.gif" height="103" width="150"
     alt="An old group photo"/></a></td>
</tr>
<tr>
    <td width="80">Aunts Betsy and Phyllis sitting on the porch.</td>
    <td><a href="auntslarge.jpg"><img src="auntssmall.gif" height="96"
width="150"
    alt="Two aunts on a porch"/></a></td>
</tr>
<tr>
    <td width="80">Don when he was a child.</td>
    <td><a href="donoldlarge.jpg"><img src="donoldsmall.gif" height="100"
width="61"
     alt="Young Don" /></a></td>
</tr>
```

Figure 13.4 shows the result.

OUTPUT

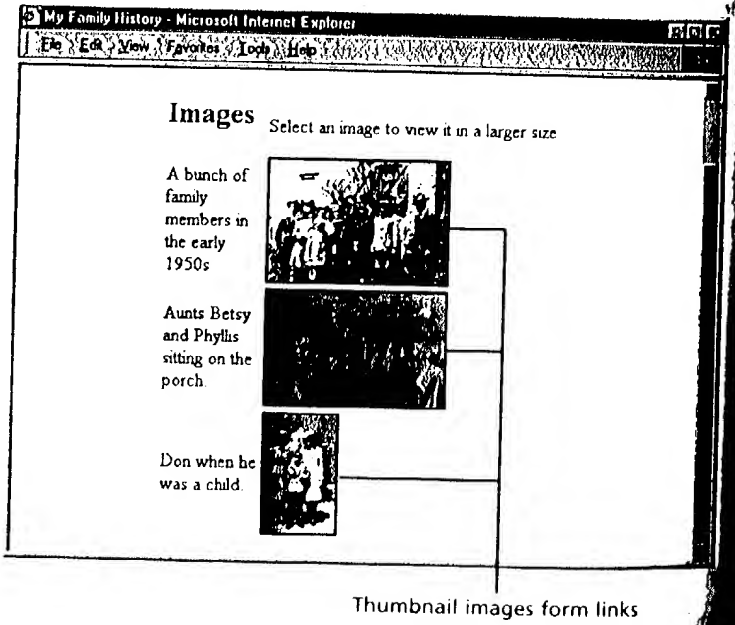


FIGURE 13.4

The image now linked to larger images.
Thumbnail images form links

▼ If I leave the archive like this, it looks nice, but I'm breaking one of my own rules: I haven't noted how large each file is. Here, you have several choices for formatting, but the easiest is to simply add the file size after the description of each picture, as follows:

INPUT

```
<tr>
    <td width="80">A bunch of family members in the early 1950s.
[103k]</td>
    <td><a href="groupoldlarge.jpg"><img src="groupoldsmall.gif"
height="103" width="150"
    alt="An old group photo"/></a></td>
</tr>
<tr>
    <td width="80">Aunts Betsy and Phyllis sitting on the porch.
[83k]</td>
    <td><a href="auntslarge.jpg"><img src="auntssmall.gif" height="96"
width="150"
    alt="Two aunts on a porch"/></a></td>
</tr>
<tr>
   <td width="80">Don when he was a child. [284k]</td>
   <td><a href="donoldlarge.jpg"><img src="donoldsmall.gif" height="100"
width="61"
    alt="Young Don" /></a></td>
</tr>
```

Figure 13.5 shows this result.

OUTPUT

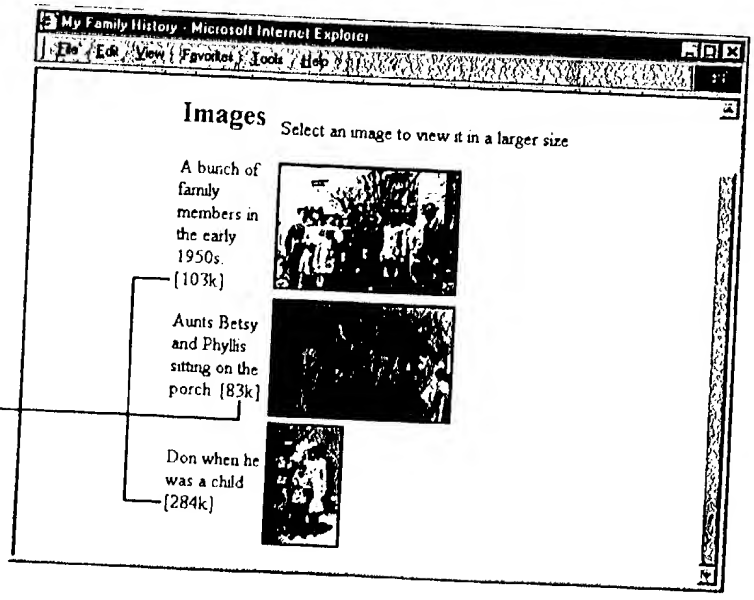


FIGURE 13.5
Adding file sizes to the description of each image enables people determine how long it will take to load the image.

▼ Now, moving on to the sound and video sections. There are two approaches to formatting these sections. You can add the material in the same table that contains the images or you can create two new tables—either way is fine. For this exercise, you are going to create new tables virtually identical to the table that held the images.

Start by adding three sound and two video files. Because the sound files can't be reduced to a simple thumbnail image, you need to describe them better in the text in the archive; however, you usually can use your video player to copy one frame of the clip and provide that as a thumbnail. Following is the code for the sound portion of your archive:

INPUT

```
<div align="center">
<table border="0">
<tr>
    <td width="150"><h2>Sound Bites</h2></td>
    <td><p>Select a sound bite to download or listen to it.</p></td>
</tr>
<tr>
    <td width="150">An oral family history describing how we survived the
tornado of 1903. [1192k]</td>
    <td><a href="tornado.wav"><img src="soundicon.gif" height="29"
width="33"</a></td>
</tr>
<tr>

    <td width="150">Don describing his first job. [1004k]</td>
    <td><a href="donjob.wav"><img src="soundicon.gif" height="29"
width="33"</a></a></td>
</tr>
<tr>
    <td width="150">Grandma Jo telling how she came to America.
[2459k]</td>
    <td><a href="jo.wav"><img src="soundicon.gif" height="29"
width="33"</a></a></td>
</tr>
</table>
</div>
```

▼ Figure 13.6 shows how the linked sounds look on the Web page.

OUTPUT

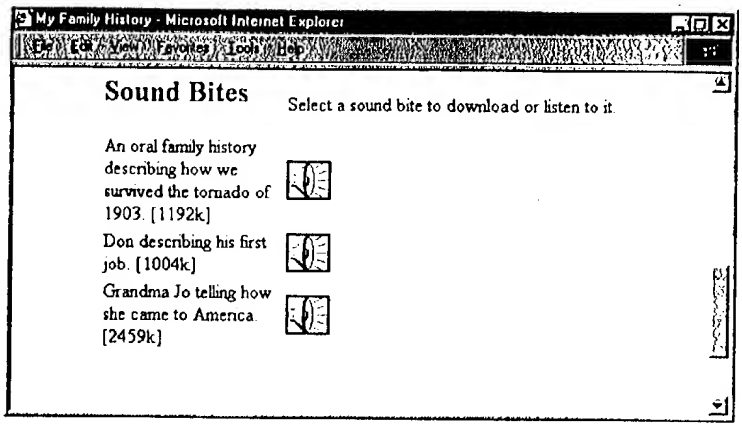


FIGURE 13.6
Linked sound files in a family history archive.

Finally, add the video clip section just as you have the previous two. It's getting easier!

INPUT

```
<div align="center">
<table border="0">
<tr>
    <td width="100"><h2>Video Clips</h2></td>
    <td><p>Select a video clip to download it.</p></td>
</tr>
<tr>
    <td width="100">A video of a family wedding. [2492k]</td>
    <td><a href="wedding.mpeg"><img src="wedding.gif" height="120"
width="180"</a></td>
</tr>
<tr>

    <td width="100">Don and Mary talking. [3614k]</td>
    <td><a href="donandmary.mpeg"><img src="donandmary.gif" height="120"
width="180"</a></td>
</tr>
</table>
</div>
```

Figure 13.7 shows how the linked videos with thumbnails look on the Web page.

▼

OUTPUT

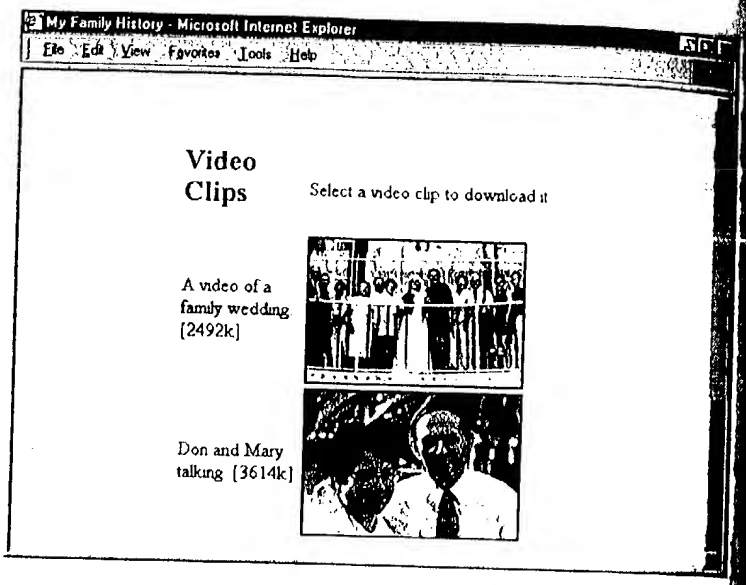


FIGURE 13.7

The video section of our multimedia archive.

Et voilá, your media archive. Creating one is simple with the combination of inline images and external files. With the use of the `alt` attribute, you can even use it reasonably well in text-only browsers.

▲

Embedding Sound and Video

Embedding sound and video is achieved through the `embed` or `object` elements. Remember, the principle behind embedding sound or video is to include it in a Web page so that it can be played as part of the page.

The `embed` element has been around for some time and is supported by both Internet Explorer and Netscape Navigator. It was created so that file types requiring plug-ins (multimedia primarily) to play could be added to Web pages. Despite this support, `embed` is not sanctioned by the World Wide Web Consortium (W3C) and can't be found in the official HTML standard. Of course, because both major browsers support the element, you can safely ignore the W3C for the time being.

The "competing" element, `object`, is officially sanctioned by the W3C, although browser support (mainly Navigator) is somewhat buggy at this time. The `object` element is supposed to provide a generic solution for embedding in Web pages all sorts of file types, from images to Java Applets to sound and video files. It is hoped that by standardizing a

"one-size-fits-all" element, you will be able to include more types of files as well as have an element ready for any future multimedia types.

Using the embed Element

Despite the fact that embed isn't in the HTML standard, Microsoft or Netscape continue supporting embed in their latest Web browsers, and there are plenty of Web pages that make use of embed.

The syntax for using embed is simple:

```
<embed attributes />
```

Note the closing tag, which is optional.

Unfortunately, despite the fact that both Internet Explorer and Navigator support embed, they share only a handful of common attributes. The flip side to that coin is that each Web browser ignores the attributes it doesn't understand, allowing you to include as many different attributes as you like. Because of this, it is best to rely on a set of attributes that will work in all cases, and use them religiously, including the others for "added value."

Let's explore the attributes you absolutely need to use the embed element.

```
<embed src="a01607av.avi" height="120" width="160" />
```

The src attribute indicates the path and name of the media file you want to embed in the Web page, while the height and width attributes set a region of the browser window aside to display the media file.

Internet Explorer and Netscape Navigator handle the height and width a bit differently. In Internet Explorer 5, the media controls of the appropriate plug-in are always displayed while Navigator seems not to want to ever display them. This causes a huge problem in that setting the height and width attribute for Internet Explorer includes the space devoted to those controls. Setting the height and width to the exact size of the video display causes the video to become "crunched" because the controls take up some of this valuable real estate. One would think that increasing the height and width would solve this problem by increasing the space and allowing the controls and the video to be displayed at their proper sizes. Because of the Netscape Navigator implementation, however, that doesn't work. Netscape Navigator expands the video to the full size of the height and width attributes and ignores the controls entirely, resulting in a video that is stretched and distorted.

Figure 13.8 and 13.9 show the problem using this code:

```
<embed src="a01607av.avi" type="video/x-msvideo" height="120" width="180" />
```

FIGURE 13.8

Internet Explorer displays the controls and sizes the overall box accordingly.

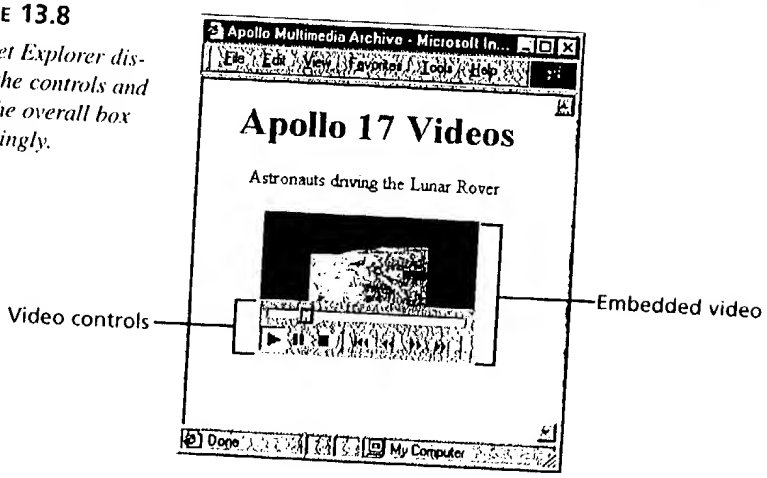
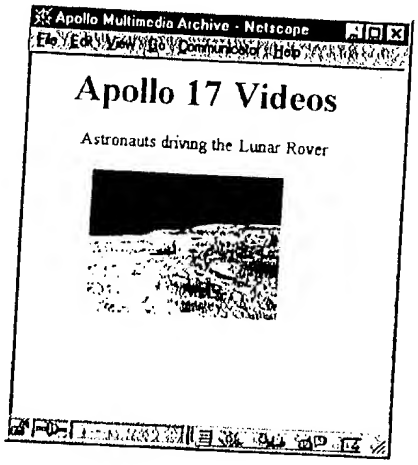


FIGURE 13.9

Netscape Navigator ignores the controls and expands the video to fit the box.



Not using the `height` and `width` attributes and allowing the browsers to handle this will work, but only in Internet Explorer. Netscape Navigator displays a small window for the plug-in, cutting off much of the video.

So what's a person to do? Apart from throwing your hands up in complete frustration, writing letters to each company telling them how mad you are, or tossing your computer out the window, if embedding video is important to you, then you can do one of the following two things:

- Pick your poison and ignore how it looks in the other browser.
- Use scripting to "sniff" for the browser type and base your use of `embed` on the visitor's Web browser.

- Use both the `object` and `embed` elements for certain file types to provide cross-browser support.

Table 13.1 summarizes the `embed` attributes supported by Internet Explorer.

Table 13.1 `embed` Attributes Used in Internet Explorer

Attribute	*Description*
`align`	Aligns the element in relation to the Web page. Allowable values are `absbottom`, `absmiddle`, `baseline`, `bottom`, `left`, `middle`, `right`, `texttop`, and `top`.
`alt`	Provides alternate text.
`class`	Sets or retrieves the class of the element.
`height`	The height of the element.
`hspace`	The horizontal margin around the element.
`id`	The ID of the element.
`name`	The name of the element.
`src`	The source of the media file.
`style`	Style Sheet declaration.
`title`	The title of the element.
`units`	Sets or retrieves the `height` or `width` units. Pixels are the default unit of measurement.
`vspace`	The vertical margin around the element.
`width`	The width of the element.

Table 13.2 summarizes the `embed` attributes supported by Netscape Navigator.

Table 13.2 `embed` Attributes Used in Netscape Navigator

Attribute	*Description*
`src`	The file location.
`type`	The MIME type of the embedded media.
`pluginspage`	A URL pointing to a Web page that has instructions for installing the required plug-in.
`pluginurl`	A URL to a Java Archive (JAR) file.
`align`	Aligns the element in relation to the Web page. Allowable values are `left`, `right`, `top`, and `bottom`.
`border`	The width of a border drawn around the element.

continues

Table 13.2 continued

Attribute	*Description*
`frameborder`	Does not draw a border around the element when set to `no`.
`height`	The height of the element.
`width`	The width of the element.
`units`	The units used to measure the `height` and `width`. Pixels are the default unit of measurement.
`hidden`	Hides the element when set to `true` and displays it when set to `false`, which is the default value.
`hspace`	The horizontal margin around the element.
`vspace`	The vertical margin around the element.
`name`	The name of the plug-in required to play the file.
`palette`	For use in Windows only. `foreground` makes the plug-in use the foreground palette, while `background` (the default) makes the plug-in use the background palette.

In addition to these attributes, additional attributes may be available for specific plug-ins, such as the Macromedia Flash Player.

Finally, you can include the `noembed` element to provide support for visitors who do not have a Web browser that can display plug-ins.

```
<noembed>This Web page requires a web browser that can display
objects.</noembed>
<embed src="a01607av.avi" height="120" width="160" />
```

Using the `object` Element

According to the World Wide Web consortium, you should use the `object` element when embedding sound and video (among other things) in Web pages. This can be problematic, as Netscape Navigator does not fully support `object` and Internet Explorer is sometime quirky about it.

To use the `object` element, start with the opening `object` tag and attributes, as follows:

```
<object data="movie.mpeg" type="application/mpeg">
```

The `data` attribute indicates the source file for your sound or video, and `type` is the MIME type of the file.

Next, include any content you want to display, such as a caption, and close the `object` element with the closing tag, as in the following:

```
<object data="movie.mpeg" type="video/mpeg">
My homemade movie.
</object>
```

You also can cascade objects so that if one cannot be displayed, the browser keeps trying down the list.

```
<object data="movie.mpeg" type="video/mpeg">
    <object data="moviesplash.gif" type="image/gif">
    </object>
My homemade movie.
</object>
```

`object` also uses the `param` element to initialize any parameters the embedded file may require. The `param` element is included in the body of the `object` element and has no closing tag, as in the following:

```
<object data="movie.mpeg" type="video/mpeg">
    <param name="height" value="120" valuetype="data">
    <param name="width" value="160" valuetype="data">
My homemade movie.
</object>
```

The preceding code sets the height and width of the embedded object to 120×160 pixels. Parameters supplied by the `param` element are dependent on the type of object you are trying to embed.

Combining `embed` and `object`

As you will see in the next few sections, you can often use `embed` and `object` simultaneously to provide support for Netscape Navigator and Internet Explorer. To do this, create the `object` element with any required parameters (the `param` element), and include the `embed` element before you enter the closing `object` tag. The following generic code snippet illustrates how:

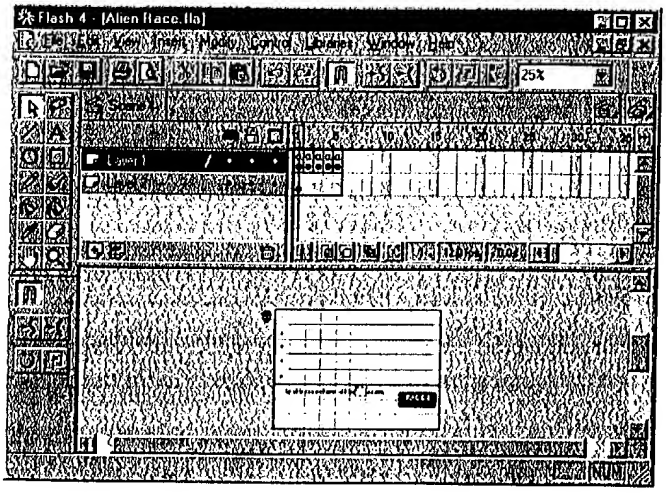
```
<object classid="value" codebase="value" height="480" width="512" name="myname">
<param name="src" value="source location" />
<embed src="filename" height="480" width="512" name="myname" />
</object>
```

When Internet Explorer loads the Web page, it will read the `object` element and use that to embed the multimedia file in the page, ignoring the `embed` element entirely. Netscape Navigator will use the `embed` element.

Embedding Flash Animations

Macromedia Flash may be one of the easier types of content to embed in a Web page because you publish the files within the Flash application. Figure 13.10 shows the Macromedia Flash interface used to create Flash files.

FIGURE 13.10
Use Macromedia Flash to create your Flash files and save them in Web pages.



Flash uses HTML templates that are modified by setting the publishing preferences. The following shows the default template:

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
 codebase="http://active.macromedia.com/flash2/cabs/swflash.cab#version=4,0,0,0"
 ID=$TI WIDTH=$WI HEIGHT=$HE>
 $PO
<EMBED $PE WIDTH=$WI HEIGHT=$HE
 TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Vers
ion=ShockwaveFlash">
</EMBED>
</OBJECT>
```

Notice that Flash uses both the `object` and `embed` elements to embed the animation in a Web page. The dollar signs ($) are variables that Flash replaces with your custom preferences when you publish your file. You can modify the HTML settings through a convenient dialog box shown in Figure 13.11.

Embedding Shockwave Animations

Unless you are using Macromedia Dreamweaver, which automatically inserts the proper HTML code into your Web page, you will have to input the code manually to embed a Shockwave file.

To embed a Shockwave file, use a combination of the `object` element (for Microsoft support) and the `embed` element (for Netscape support). You should use both to ensure maximum compatibility.

FIGURE 13.11
Flash enables you publish your anim tions using an HT template that is cu tomizable from thi dialog box.

For most Sl your own v: require diff retailer or d

The source

```
<OBJECT CLA
CODEBASE="h
ion=7,0,0,0
<PARAM NAME
<EMBED SRC=
</OBJECT>
```

Table 13.3 li

Table 13.3 obje

Attributes
`classid`
`codebase`

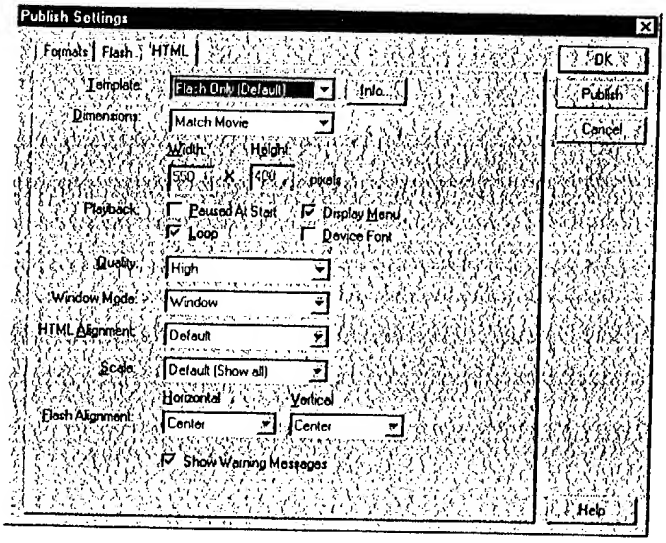


FIGURE 13.11

Flash enables you to publish your animations using an HTML template that is customizable from this dialog box.

For most Shockwave applications, you can enter the code as shown and simply substitute your own values for the location and size of the movie. Other embedded objects may require different information. You should always consult any information from the object retailer or developer for the exact parameters.

The source code for embedding Shockwave files takes the following general form:

```
<OBJECT CLASSID="clsid:166B1BCA-3F9C-11CF-8075-44553540000"
CODEBASE="http://download.macromedia.com/pub/shockwave/cabs/director/sw.cab#vers
ion=7,0,0,0" WIDTH="512" HEIGHT="480" NAME="MovieName">
<PARAM NAME="SRC" VALUE="MYMOVIE.DCR">
<EMBED SRC="MYMOVIE.DCR" HEIGHT=480 WIDTH=512 NAME="MovieName">
</OBJECT>
```

Table 13.3 lists the attributes you can use in the `object` element.

Table 13.3 `object` Attributes

Attributes	*Description*
`classid`	The universal class identifier for the Shockwave ActiveX Control. It must be set to the following value: `clsid:166B1BCA-3F9C-11CF-8075-444553540000`
`codebase`	Specifies the download location of the Shockwave control, if not currently installed. It must be set to the following value: `http://download.macromedia.com/pub/shockwave/cabs/director/sw.cab#version=7,0,0`

continues

Table 13.3 continued

Attributes	*Description*
`width`	The width of the Shockwave file (in pixels).
`height`	The height of the Shockwave file (in pixels).
`name`	The name of the Shockwave movie (text).
`src`	This attribute is used in a `param` element within the `object` element. The `name` should be `src` and the `value` points to the URL of the movie. Following is the code: `<param name="src" value="URL">`
`bgcolor`	The background color of box that holds the movie before it appears (hexadecimal number).
`swmodifyreport`	If `true`, it removes the `src` URL from Shockwave statistics collection (`true`).

Table 13.4 lists the `embed` attributes.

Table 13.4 `embed` Attributes

Attribute	*Description*
`width`	The width of the Shockwave file (in pixels).
`height`	The height of the Shockwave file (in pixels).
`name`	The name of the Shockwave movie (text).
`src`	The location and name of the movie (URL).
`pluginspage`	Applies to Netscape Navigator only. The URL of the Shockwave Plug-in. The value should be as follows: `http://www.macromedia.com/shockwave`
`bgcolor`	The background color of box that holds the movie before it appears (hexadecimal number).
`swmodifyreport`	If `true`, it removes the `src` URL from Shockwave statistics collection (`true`).

Shockwave files are created in Macromedia Director (see Figure 13.12).

Embedding RealAudio and RealVideo

RealNetworks RealAudio and RealVideo files also use `object` and `embed`. Following is the syntax for including the files:

```
<object id="RVOCX" classid="clsid:CFCDAA03-8BE4-11cf-B84B-0020AFBBCCFA"
width="300" height="134">
Optional parameters
```

```
<embed src="source" width="value" height="value" />
<noembed><a href="download page">Play with RealPlayer.</a></noembed>
</object>
```

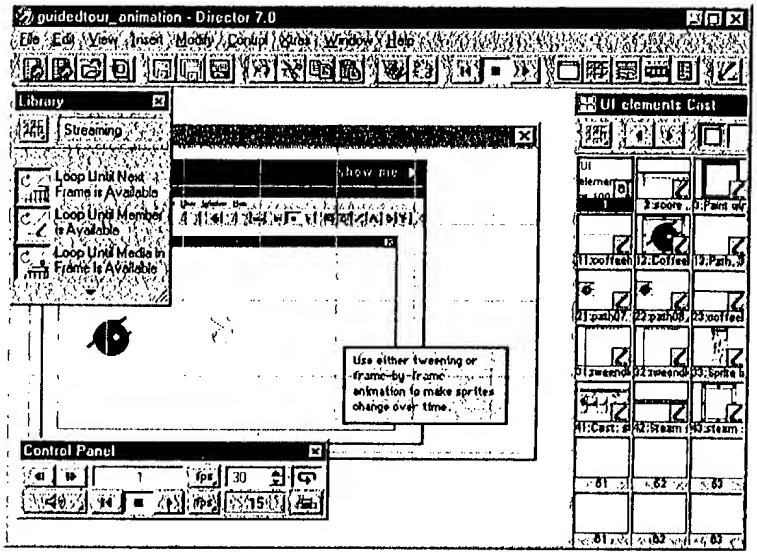


FIGURE 13.12
Macromedia Director 7 can save movies as Shockwave files.

Table 13.5 lists the available attributes for `embed` and the parameters (`<param name="name" value="value" />`) for the `object` element.

Table 13.5 `embed` Attributes and `object` Parameters

Attribute/Parameter	*Description*
`autostart`	Sets automatic playback (`true` or `false`).
`backgroundcolor`	Sets background color (hexadecimal color value or name).
`center`	Centers clip in window (`true` or `false`).
`console`	Links multiple controls (`yes`, `name`, `_master`, or `_unique`).
`controls`	Adds RealPlayer controls (control name).
`height`	Sets window or control height (in pixels or percentage).
`loop`	Loops clips indefinitely (`true` or `false`).
`maintainaspect`	Preserves image aspect ratio (`true` or `false`).
`nojava`	Prevents the Java Virtual Machine from starting (`true` or `false`).
`nolabels`	Suppresses presentation information (`true` or `false`).
`nologo`	Suppresses RealLogo (`true` or `false`).

continues

Table 13.5 continued

Attribute/Parameter	*Description*
`numloop`	Loops clip a given number of times (number).
`region`	Ties clip to SMIL region (SMIL region).
`shuffle`	Randomizes playback (`true` or `false`).
`src`	Specifies source clip (URL).
`width`	Sets window or control width (in pixels or percentage).

You also can use the Web Page Wizard in RealProducer Plus to automatically create Web pages with embedded RealMedia.

Multimedia Techniques Using Microsoft Internet Explorer

Microsoft Internet Explorer offers a few unique capabilities worth mentioning: background sounds and inline video. Note, however, that while Netscape Navigator does not support either of these two techniques, you can safely include them in your Web pages. Navigator will ignore background sounds and you can code inline video in such a way that Navigator will display a static image in place of a video.

Including Background Sounds

Internet Explorer has an element that loads and plays audio files in the background. These sound files load when the page loads into the reader's Web browser and play without the user having to do anything special like starting the audio playback manually. Because no visual effect is created, there will be no indication that a sound is playing unless the users have a sound card and the volume is turned up. To add an embedded background sound to a page, use the `bgsound` element:

```
<bgsound src="ElevatorMusic.wav" />
```

Use the `loop` attribute to repeat the sound multiple times. If the value of `loop` is a number, the sound is played that number of times. If `loop` is `-1` or `infinite`, the sound will repeat continually, until the reader leaves the page.

```
<bgsound src="ElevatorMusic.wav" loop="-1" />
```

Explorer supports three different formats for inline sounds: Sun's popular AU format, Windows `WAV` files, and `MIDI` files with a `MID` extension.

As with the inline video extensions, covered in the following section, the `bgsound` element is not supported in Netscape's browsers.

PART 5

DAY 15

Using Dynamic HTML

In yesterday's lesson, you learned how to create powerful HTML forms that enable you to gather information from people visiting your Web site. Forms, in one "form" or another, have been around since the virtual birth of HTML and are pretty compatible among browsers. You didn't really need to worry too much about who could or couldn't see your site correctly.

Today's lesson departs from the relatively smooth waters of forms and sets sail toward the rough seas of Dynamic HTML. Batten down the hatches!

Dynamic HTML offers you something unique: the capability to make changes in Web pages on-the-fly. Using conventional HTML, Web pages are loaded by a browser and just sit there until you click a link or interact with forms. That's pretty simple, but can be static and boring.

People have improved upon static Web pages by using a variety of technologies outside the realm of HTML. CGI has been a longtime method of providing interactivity in Web pages. You can embed Shockwave and Flash to create animation and interactivity, and use Java applets or ActiveX controls to provide application-like functions; however, these methods rely on browser plug-ins or virtual machines (which can be messy), and can lead to longer download times.

Dynamic HTML is different. *DHTML*, as it is commonly known, enables you to create Web pages that look, feel, and act a lot like the other programs you use on your computer—using the Web browser as an interface, having to rely on external programming solutions. Can you see the promise of that?

I hope you do, because you'll need that enthusiasm to come to grips with the darker side of DHTML. This side is sometimes, quite frankly, a mess. If you want to appeal to the largest possible audience (within realistic limitations), you must account for differences in Web browsers and how they implement a variety of Web technologies including HTML, Cascading Style Sheets, and scripting.

Today is a "full plate" of information, so let's get right to it. In this lesson, you will learn about the following:

- Defining Dynamic HTML and the technologies that make it possible
- The basics of JavaScript
- Understanding what Document Object Models are
- Creating cross-browser routines with DHTML, such as sniffing for browsers and DOM references
- Creating expandable menus and a neat tic-tac-toe game with DHTML

What Exactly Is Dynamic HTML?

Simply put, Dynamic HTML uses normal HTML elements to create a Web page that relies on style sheets for element formatting, positioning, and scripting to *dynamically* change either HTML content, style, or positioning, without having to *re-download* the page from the server. Dynamic HTML isn't a "thing" by itself, but a collection of technologies working together to achieve interactive effects. So what can you do with DHTML? The possibilities are endless! Just to whet your appetite, here are a few of them:

- Move objects around the Web page
- Show or hide elements
- Create lists that expand or contract when you select an item
- Dynamically alter the color and size of Web content
- Provide drag-and-drop functionality similar to that used by modern graphical operating systems (such as Windows and Macintosh)

Dynamic HTML was born with the advent of the "fours." The World Wide Web Consortium developed HTML 4 and released the official Recommendation at about

15

same time Microsoft and Netscape released their competing Web browsers, Internet Explorer 4 and Navigator 4. There was a flurry of activity surrounding all this. HTML 4 promoted several new features to better integrate itself with Cascading Style Sheets and respond to user events. Microsoft and Netscape were both competing hard for market-share in the browser war, and the result of it all was a drive for something different: a level of user interactivity and "dynamism" in Web pages that previously was impossible without resorting to external programming. DHTML was born.

New Term *Dynamic HTML* (DHTML) is a collection of Web technologies that increases the interactivity and dynamism of Web pages without resorting to external objects or programs.

For DHTML to work, it requires the following three key technologies supported by the Web browser:

- HTML
- Style sheets
- Scripting

It's obvious that you need to use HTML. Remember, however—HTML 4 has new elements, such as `div` and `span`, and new attributes, such as `id` and `style`, that enable you to structure and format your content in ways that promote a tighter integration with style sheets and manipulation by scripting. The darker side of this is that even today, not all the leading manufacturers of Web browsers consistently implement HTML 4 in their Web browsers. This will be a common theme throughout this chapter.

Note

> The greatest challenge of Dynamic HTML is to overcome these inconsistencies—not just in HTML, but in all DHTML technologies—and create true, cross-browser dynamic Web pages. It *can* be done, but it takes more effort.

Style sheets are wonderful, once you know how to use them, and are a critical component of DHTML. Although DHTML technically isn't dependent on any one type of style sheet, the official W3C style sheet technology, Cascading Style Sheets (CSS), is the standard. CSS level 2 (CSS2) enables you to format elements on your Web pages using properties, such as font, color, and spacing, as well as positioning items on the Web page. To review CSS, refer to Day 10, "HTML and Style Sheets."

DHTML takes style to a new level. Rather than creating a style rule or positioning an element on the page and then forgetting about it, you can use DHTML to dynamically alter the visual style or position of your elements. As with HTML, there are problems

with how Internet Explorer and Netscape Navigator implement CSS. Not only have neither *fully* implemented the specification, but each browser has implemented portions differently. Argh!

Finally, scripting is a sort of "glue" that holds together everything in DHTML. Scripting provides the dynamic nature of DHTML because scripts can run while a page loads, in response to a user action, or even when the user leaves your site.

Caution

If you've caught onto the pattern here, "inconsistent implementation across Web browsers," you should know that this makes referring to and relying on the "official" specification of any one technology dangerous. You should always read any browser-specific documentation you can find to ensure that the Web browsers you are targeting support what you are attempting to accomplish.

The scripting "lingua franca" of today is JavaScript. JavaScript was the first scripting language to be used with the Web, and currently enjoys the most widespread support across different Web browsers. Although you can create DHTML using other scripts, such as VBScript, I recommend always using JavaScript unless you are in a Microsoft-only environment (which I have yet to see in reality).

As you will see when we get to the DHTML examples, a large part of DHTML involves creating scripts that change elements' style, position, and perform other calculations to provide the "D" in DHTML. For this reason, I have included a short introduction to JavaScript, which you will read a bit later. Although you won't learn everything there is to know about JavaScript, you should at least be able to follow along with the examples.

Two final notes about what DHTML actually is. DHTML (the scripting part of it) relies on something called a *Document Object Model* (DOM) to identify, create, and manipulate objects in a Web page. For example, you have to be able to identify an element, such as an image, in order to manipulate it with a script. Likewise, you must be able to identify an element's style in order to change it. This is what the DOM does. It provides a bridge between the content of the Web page and scripts. Although the W3C currently is working on official DOM specifications, each Web browser has a unique DOM; therefore, the DOMs are covered in a separate section later in this chapter.

Finally, DHTML relies on *event handling* to track the actions of the Web browser and user. When the page loads, the `onload` event is triggered. Likewise, a when visitor click a button in a form, several events might be triggered that you, the DHTML author, can use to run scripts. As with everything else in DHTML, Microsoft Internet Explorer and

Netscape Navigator handle events in different fashion. Event handling is covered in more depth later on.

Learning JavaScript

JavaScript is a scripting language originally developed by Netscape and Sun Microsystems (the people who created Java), and was created to solve some of the shortcomings of HTML. Namely, HTML is a document *formatting* language not capable of performing "programming" tasks. Take the following HTML code, for example:

```
<img src="myImage.gif" height="100" width="300" />
```

In this case, the HTML code simply is telling the browser to display an image called `myImage.gif` on the page in an area 100 pixels high and 300 pixels wide. This is a pretty severe limitation when you think about it. Aside from hyperlinks and form controls, you can't manipulate information, data, objects, or respond to user events.

NEW TERM *JavaScript* is a scripting language that allows you to write scripts, or small non-compiled programs, that are run by a Web browser from within a Web page.

Scripting languages, in general, and JavaScript, in particular, perform completely a different role than HTML. By their very nature, they are *programming* languages. This means that scripts can perform computations, manipulate objects, and respond to a wide variety of user events. Scripts, for example, perform tasks such as validating HTML form input, responding to mouse and keyboard actions, and dynamically change the position and style of HTML elements.

Unlike formal programming languages, such as C, C++, or Java, scripting languages are not compiled into machine-readable code prior to being executed. They are written in normal text and interpreted by a host rather than executed directly by a CPU. In the case of Web pages, the Web browser acts as the host and interprets scripts within (or referenced by) the Web page while it is loading.

Note

This is an important point worth repeating. Just as some older browsers cannot display many HTML 4 elements, not all browsers can interpret the latest version of JavaScript. Remember, it is the browser that is acting as the script host. You can't force a browser to run scripts it doesn't understand.

JavaScript Basics

Unfortunately, I can't teach you everything there is to know about JavaScript in such a short space. I will, however, try to give you an understanding of how JavaScript works so

that you can look through the code in this lesson, and begin to understand what's being done.

Fundamentally, scripts are nothing but a series of statements that tell the script host (the Web browser) what to do. These statements can be simple commands setting variable values, such as the following:

```
var x = 14;
```

or more like the following complex multiline statements, that define functions:

```
function init() {
    if (x == 14) {
        alert(x);
    }
    else {
        alert("X doesn't equal 14");
    }
}
```

Notice that each statement ends in a semicolon (`;`) and that statements can be grouped together into blocks with curly braces `{}`. The preceding code uses braces to group statements into a function called `init()`. The `if` and `else` statements each use braces to contain the statements that apply to them.

You can include information within JavaScript scripts that will not be interpreted. These are called *comments*, and are used to explain portions of the script. You create single-line comments by using two forward slashes, `//`. You can place them at the beginning of a line, such as:

```
// This variable is used to hold temporary information
var x;
```

or you can place them at the end of a statement, such as:

```
var x; // Temporary variable
```

The script interpreter ignores everything after the double forward slashes, so be careful not to comment-out any important code!

You can use single-line comments to create a large comment block, such as:

```
// Script created on 9 Oct 99
// Copyright Sams Publishing
// All rights reserved
```

or use a special multiline comment. Multiline comments are created by entering a forward slash followed by an asterisk (`/*`) on the first line of the comment, and end with an asterisk and another forward slash (`*/`). The following example illustrates that type of multiline comment:

```
/* Script created on 9 Oct 99
   Copyright Sams Publishing
   All rights reserved */
```

Variables are one of the most convenient programming tools. They enable you to assign values to placeholders so that you can manipulate the data by referring to the placeholder. To declare a variable, all you need to do is enter `var` followed by the name of your placeholder, as in the following:

```
var myVariable;
```

Note

JavaScript is case-sensitive. This means that `myVariable` is different from `myvariable` and `Myvariable`.

If you want to declare a variable and immediately store a value in it, the code is a simple as this:

```
var myVariable = 1999;
```

JavaScript also contains operators that enable you to perform mathematical calculations and comparisons. Table 15.1 lists the most common operators and what they do.

Table 15.1 Common Computational and Comparative Operators

Computational		*Comparative*	
Symbol	*Description*	*Symbol*	*Description*
+	Add	=	Assigns values
-	Subtract	==	Is equal to
*	Multiply	<	Less than
/	Divide	>	Greater than
++	Increment (+1)	<=	Less than or equal
--	Decrement (-1)	>=	Greater than or equal
		!=	Is not equal
		&&	Logical AND
		||	Logical OR

The following code, for example, declares three variables, assigns values to two, and then assigns the sum of the first two to the third:

```
var xPos = 142;
var yPos = 15;
var Pos;
Pos = xPos + yPos;
```

To increment or decrement a value, use those operators with the value or variable, as in the following:

```
Pos++;
```

This statement adds one to the current value of Pos.

To compare values, use the logical operators.

```
if (Pos == 150) {
    anotherFunction();
}
```

This brings me to my next point: *conditional branching*. The `if` statement is one of the most common conditional statements you can find in JavaScript. Use the `if` statement to compare one value to another (or perform a whole group of comparisons), and then, based on the result, take action. The previous code compared the variable `Pos` to `150`. If `Pos` equals `150`, then the code within the braces is executed. If `Pos` does not equal `150`, the script continues on as if nothing had happened. If you want to try to catch all the results, use multiple `if` statements or include the `else` statement, as in the following code:

```
if ((Pos >= 150) && (Pos <=200)) {
    functionOne();
}
else {
    functionTwo();
}
```

This loop examines the variable `Pos` against two values: `150` and `200`. If `Pos` is greater than or equal to `150` *and* less than or equal to `200` (in other words, if `Pos` is from 150–200), then the code branches to the function titled `functionOne`. If the logical comparison is not true, the `else` block is executed and `functionTwo` is called.

NEW TERM *Conditional branching* creates two or more possible execution paths for a script to take. Conditional statements such as `(X > Y)` are used to test values and variables and determine the actual path.

When performing calculations or comparisons, parentheses group values or variables and determine the order in which they are interpreted. The following line of code forms a statement where `Pos >= 150` and `Pos <=200` are each interpreted first, and then the result of each is compared with the logical `AND` operator.

```
((Pos >= 150) && (Pos <= 200))
```

Likewise,

```
(((156 * 4 / 24) + 99) * 24)
```

is interpreted with the help of parentheses. In this case, 156 is multiplied by 4 and the result is divided by 24. That result is added to 99, and finally multiplied by 24. The final calculation results in 3000 (which, by completely accidental happenstance, is a nice round number).

Note

When confronted by multiple levels of parenthetical expressions, work from the inside out. You also should check to make sure that you always have the same number of left parentheses as you do right parentheses.

In addition to performing this task, you also use parentheses when creating a function or calling it from within the script. The parentheses contain parameters or arguments that the function uses to perform its task. The following code illustrates a simple function designed to create an alert box that displays the value of the `user` variable, which, in this case, is `Laura`.

```
var user = "Laura";
function alertUser(user) {
    alert(user);
}
```

To call the function, simply enter the function name with the parameter `user` enclosed in parentheses, as in the following:

```
alertUser(user);
```

Many functions do not have parameters, but they all must have the parentheses. For example, a sample `init` function listed here takes no parameters when called:

```
function init() {
    createImageArray();
    animateTitle();
}
```

This brings me to my next point: *functions*. Normally, scripts execute from top to bottom, in the order the statements are entered. Functions, however, break this flow and are blocks of code that are only executed when they are called from other parts of the script. Consider the example just listed. The `init` function stands apart from the script and does not execute until called. Therefore, when writing scripts, use functions to control the flow of execution and respond to unique events.

New Term A *function* is a block of code that is set apart from the script's normal flow of execution. Functions run only when called from other parts of the script or HTML document.

If you haven't guessed by now, you can create functions by using the keyword `function`, followed by the function name. Remember to use the left and right parentheses to enclose any arguments, even if there are none. Following this, enter a left brace, and then input any statements you want the function to execute. To close the function, use a right brace.

You can include conditional statements, such as `if`, and loops within functions, like this:

```
function determinePosition() {
    if ((Pos >= 150) && (Pos <=200)) {
        functionOne();
    }
    else {
        functionTwo();
    }
}
```

Just make sure you have an equal number of opposing braces within the function. In the preceding listing, the `determinePosition` function has a set of braces at the top and bottom. Within the function, the `if` and `else` conditional statements each have a set of braces. It all adds up. There are three left braces and three right braces. If there weren't, you would receive an error when the script was executed.

Thus far, you've learned the basic syntax of JavaScript and some details of the language. Apart from adding a few values together, however, you really don't know how to do much. The missing link is, drum roll please, an *object*. Objects are simply collections of *properties* and *methods*. Think of them as containers that possess information, as well as the means to manipulate it.

NEW TERM *Objects* are collections of properties and methods, holding information and the means to manipulate it.

Properties describe the object, and methods provide the means to manipulate it with your script. Suppose that you have an object called `Car`. Its properties might be the `color`, `year`, and `value`. You might have a method that calculates its current value based on the depreciation each year. To refer to the object, enter the object name followed by a period and then include a property. This example sets the `Car` object's color to `blue`.

```
Car.color = "Blue";
```

To invoke a method, simply add the method after the object and property, as in the following code:

```
Car.value.calculate();
```

Because we're discussing scripting for the Web, most of the objects contain properties and methods that relate to the Web browser and HTML document. One of the most

important objects is the `document` object. This object contains information about the HTML document loaded into the Web browser, has methods to respond to user events, and can perform other tasks.

Unfortunately, here's where it gets a bit more complicated. Each Web browser has defined a different set objects that may have different properties and methods. In addition, each browser has a set of *event handlers* that are used to respond to user actions, such as clicking the mouse or pressing a key.

Note

> In Netscape Navigator, not all event handlers are available for all objects. You should consult Netscape's JavaScript documentation to see which events can be used for certain objects.

Because objects and events are somewhat problematic across the different browser versions, I will cover those subjects in more depth in following sections. First, however, examine how to integrate your scripts into HTML documents.

Integrating Scripts with HTML

JavaScript, of course, isn't HTML. They are entirely two different languages that perform completely different functions. You can't just throw script statements in your HTML document willy-nilly and expect the browser to automatically know what to do. The first thing you should do is specify a default scripting language in the `head` of the document, using the `meta` tag, employing the following statement:

```
<meta http-equiv="Content-Script-Type" content="text/javascript" />
```

Although using the `meta` tag to perform this function is optional, this statement ensures that the Web browser knows that you are using JavaScript, and, unless told differently, all scripts will be interpreted as such. That distinction is very important because you don't want to leave any doubt for the browser. Now on to including the scripts!

You can include JavaScript scripts in your HTML documents using any of the following three methods:

- Include inline scripts in response to intrinsic events
- Put scripts in the document `head` or `body`
- Link to external script files

Intrinsic events occur when something happens to the Web page. The `body` of the HTML document, for example, has an intrinsic event called `onload` that occurs when the document is loaded into the Web browser. These events are used as attributes to HTML

elements. The value of the attribute can be a script statement or a series of script statements. The following listing, for example, assigns the `onload` event to the `body` element and then executes a single statement that creates an alert box with the included text.

```
<body onload="alert('The page has loaded.')">
```

Note that inline script statements that need to use double quotation marks must use single quotation marks in this setting. The first double quotation mark begins the attribute value, and the next one it runs across will terminate the value. If you were to mistakenly enter the following code, the script would end at `alert(`, which obviously is something you don't want.

```
<body onload="alert("The page has loaded.")">
```

Caution

When it comes to intrinsic events, don't be fooled by the official HTML specification. Although the W3C lists a multitude of events that apply to certain elements, it is up to the Web browser to implement this functionality. Netscape Navigator in particular does not support intrinsic events "across the board" as Internet Explorer does. This fact makes DHTML harder when coding for Navigator.

You also can use multiple script statements inline when responding to an intrinsic event. Consider the following code:

```
<body onload="var message = 'The page has loaded'; alert(message)">
```

I have two statements within this small script that respond to the `onload` event. The first statement declares a variable and assigns a string to it. The second displays an alert box that contains the message.

The second method for including scripts is to put them in the HTML document `head` or `body`. To include a script, use the `script` element, and declare the type of script with the `type` and/or the `language` attribute in the opening `script` tag. Both attributes perform the same function in that they tell the browser which scripting language you are using. Although `type` is the official standard, `language` is more commonly used. Because it doesn't do any harm, I tend to include both.

```
<script type="text/javascript" language="javascript">
```

After the opening tag, enter any script statements you have created, and then close the `script` element with an end tag.

```
<head>
<title>Sample Script</head>
    <meta http-equiv="Content-Script-Type" content="text/javascript" />
    <script type="text/javascript" language="javascript">
        var message="Hello World";
        alert(message);
    </script>
</head>
```

Although the previous script will run just fine in browsers that are "script compatible," it is common practice to comment out the script statements with HTML comment tags, as follows:

```
<script type="text/javascript" language="javascript">
<!--  Hide JavaScript from older browsers
    var message="Hello World";
    alert(message);
// stop hiding the script -->
</script>
```

Notice the last line of the script. I have included a JavaScript comment to ensure that the JavaScript interpreter does not think it is a line of script. If I ended the script with the following HTML comment, the browser would try to interpret it as JavaScript:

```
stop hiding -->
```

Another technique to ensure backward compatibility is to use the `noscript` element. This element enables you to include alternate content for browsers that don't support scripting. Begin with the opening `noscript` tag, and then enter your alternate content. Close the `noscript` element with an end tag, as in the following:

```
<script type="text/javascript" language="javascript">
<!--  Hide JavaScript from older browsers
    var message="Hello World";
    alert(message);
// stop hiding the script -->
</script>
<noscript>
    <p>You have a browser that is not compatible with scripting.</p>
</noscript>
```

As I stated earlier, you can include scripts in the document `head` or `body`. It is important to note that the order in which the scripts appear is the order that they execute. Therefore, a script that appears in the document `head` will run before one in the `body`. If you are using functions to control the flow of your script, you can effectively bypass this rule and control script execution based on how you intend it to work.

Finally, you can link to one or more external scripts (saved with a `.js` extension) by using the `src` attribute within the `script` element. The following example illustrates linking to two external scripts:

```
<script type="text/javascript" language="javascript" src="detect.js"></script>
<script type="text/javascript" language="javascript" src="animate.js"></script>
```

By linking to external scripts you can more easily manage common script routines that you use across multiple Web pages.

Now you know how to include scripts in your Web pages! To conclude this section, you should know that JavaScript is currently at version 1.3, but version 1.4 soon will arrive. Table 15.2 summarizes the JavaScript versions supported by Netscape Navigator and Microsoft Internet Explorer.

Table 15.2 JavaScript Versions Supported by Netscape Navigator and Microsoft Internet Explorer

JavaScript	*Navigator*	*Internet Explorer*
1.0	2.0*x*	3.0*x*
1.1	3.0*x*	4.0*x*
1.2	4.0-4.05	4.0*x*
1.3	4.06-4.61	5.0*x*
1.4	TBD	TBD

Note

Microsoft Internet Explorer technically does not interpret JavaScript, but a Microsoft script implementation that is based on JavaScript, called *JScript*, does. Don't let this confuse you! You should continue to write your scripts based on the JavaScript documentation.

Using Document Object Models

As mentioned earlier, Microsoft Internet Explorer and Netscape Navigator each have different document object models. Each object in the Web page (and many in the Web browser) contain different properties (that describe them), methods (that manipulate them), and event handlers (to respond to user actions). Next, you briefly will review each browser's model in order to arrive at some similarities.

Note

Fully realizing the capabilities of either DOM requires a depth of knowledge beyond the scope of this book. But, as with other Web-related code, you can learn a lot by cutting and pasting scripts and taking the time to study their results.

The Navigator DOM

Navigator's DOM is synonymous with the DOM presented in the official JavaScript documentation. This can be incredibly confusing when attempting to create cross-browser DHTML because while Internet Explorer interprets JavaScript, it does so in a different DOM context: its own. In other words, be careful when reading official JavaScript documentation. The information contained within applies only to Netscape Navigator.

Aside from a few top-level, predefined JavaScript objects, such as `screen`, the Navigator DOM is structured like a tree, with the topmost object called `window`. The `window` object represents the Web browser window or a frame within the browser. As such, the properties and events of the `window` object pertain to the Web browser.

Below the `window` object are several "sub-objects," with some of these branches branching themselves into more objects. The object directly under `window` that you are most interested in for your purposes today is the `document` object. The `document` object is created in Navigator by the HTML `body` tag, so you don't have to worry about creating it yourself.

Beneath the `document` object can be a plethora of different objects, each with its own set of properties and methods. Most of the objects you will use are HTML elements, such as images, style sheet objects, and form controls.

You continually will reference the `document` object and its descendants in DHTML, so it's important to learn how to reference them. When you call on an object in JavaScript, use dotted notation to refer to each object in the DOM hierarchy until you reach your destination. Following, for example, is how a reference to an image named `myimage` would appear:

```
document.myimage
```

To refer to the properties of the image, simply tack the property onto the end of that statement, as follows:

```
document.myimage.src
```

Perform a simple experiment. Create a Web page that has one image. (You can name it as I have, or use another name. Just be sure to substitute the correct value for the `src` attribute.) The image is located in the `body` of your HTML document. Just below the image is a script that will create an alert box that displays the `width` attribute of the image. For this example, it is important to place the script *after* the image, because the image needs to be loaded in order for the object to be created.

INPUT

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>Netscape Navigator DOM Test</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>

<img id="barnimage" name="barnimage" src="barn.gif" height="100"
width="150" />

<script language="javascript" type="text/javascript">
<!-- Hide JavaScript
alert(document.barnimage.width);
// end hide JavaScript -->
</script>

</body>
</html>
```

Figure 15.1 shows the resulting alert box generated by Netscape Navigator. Notice that the image has yet to appear before the alert box is shown.

OUTPUT

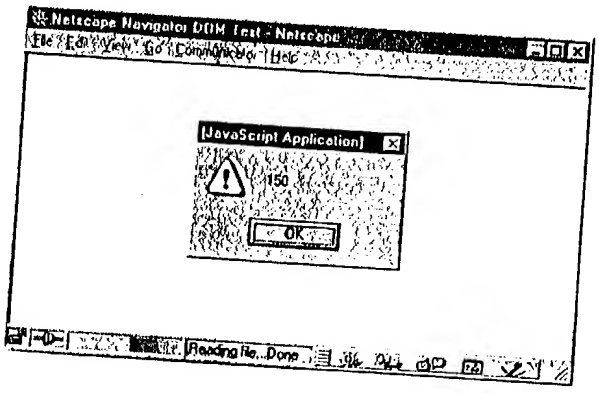


FIGURE 15.1

Accessing the properties of an image through JavaScript and Navigator's DOM.

Accessing other properties of the image is as easy as changing `width` to some another property, such as `src`.

To use a method of an object, simply use the method in place of the property, including any values that are required by the method. The following code, for example, uses the `write` method of the `document` object to write to the browser window:

```
document.write("Hello world!")
```

Some properties, such as the height of an image, are read-only. This means that you can view the value of the property; however, you cannot change it. Others, especially CSS properties (which are descendants of the object itself), are capable of being modified. To modify an object property, enter the object and property name as you normally would refer to it; however, include an equal sign, and then enter the new value. The following statement, for example, modifies the position of the image object `barnimage` to `150` pixels from the top of the window or previous element, depending on whether the image was positioned absolutely or relatively.

```
document.barnimage.top = "150"
```

One final point about the Navigator DOM is that the objects "under" an object are stored in an *array*. All image objects on a Web page, for example, are stored in the `document.images` property, as an array. The first image is stored in position `0`, with each image following the first. To access the first image in the array, use the following syntax:

```
document.images[0].width
```

Substituting this statement in the preceding image example, which created an alert box, would yield the same value: `150`.

The Internet Explorer DOM

The Internet Explorer (versions 4 and 5) DOM is similar to Navigator's DOM in that each browser creates objects with properties, methods, and events. The differences lie, however, in how they are organized and how you reference them in your scripts.

Internet Explorer has a `document` object much like Navigator's; however, underneath the `document` object, everything is contained in the `all` collection, which literally contains every HTML element on the Web page. When you need to reference an image in Internet Explorer, therefore, the statement would be as follows:

```
document.all.myimage
```

To access a property of method, add it to the end of the previous statement, just as you did with Navigator:

```
document.all.myimage.src
```

Style objects are treated differently in Internet Explorer. All references to style are handled by entering `.style` after the object name, and then entering the style property.

```
document.all.barnimage.style.top
```

You finish off the DOM discussion with the same image example for Internet Explorer that you used for Netscape Navigator.

INPUT

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/transitional.dtd">
<html>
<head>
<title>Internet Explorer DOM Test</title>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>

<img id="barnimage" name="barnimage" src="barn.gif" height="100"
width="150" />

<script language="javascript" type="text/javascript">
<!-- Hide JavaScript
alert(document.all.barnimage.width);
// end hide JavaScript -->
</script>

</body>
</html>
```

Figure 15.2 shows the resulting alert box. Notice that in Internet Explorer, the image appears before the alert box is generated.

OUTPUT

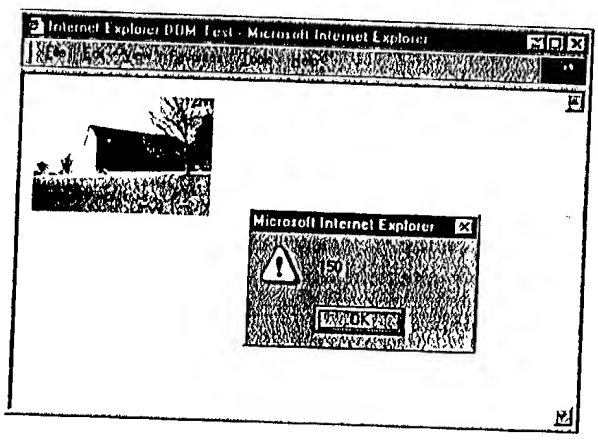


FIGURE 15.2

Accessing the properties of an image through JavaScript and Internet Explorer's DOM.

Handling Events

Events occur when the Web browser or user performs an action. The significance of events is that you can create scripts to respond to them. Unfortunately, there are wide differences in the ways Navigator and Internet Explorer track and detect events. In general, Navigator tends to be much more restrictive in the HTML elements that automatically trigger intrinsic events. You can assign many events to almost all elements in Internet Explorer.

Note

Because of the number of events and the differences in implementation, it is impossible to completely cover all contingencies in this book. For complete details, refer to Navigator- and Internet Explorer-specific documentation.

Another difference in event handling has to do with where the event is first detected, and which direction it travels (up or down) the document object hierarchy. Navigator uses a "trickle down" event model, whereby events are detected at the highest level in the document object model (the `window` object), and travel down the object model tree until an event handler responds to the event. Internet Explorer, not surprisingly, handles events in a completely opposite manner. Rather than trickling down, events in Internet Explorer "bubble up." The event first is detected by the element that generated the event (such as an image or `div` element). The event then moves up the document object chain, until handled or cancelled.

The practical upshot of this is that coding for events is pretty easy in Internet Explorer, but much more difficult in Navigator.

Coping with Reality: Cross-Browser DHTML Techniques

The majority of the chapter thus far has been devoted to understanding the technologies behind DHTML and how to use them separately. It's time to start creating real DHTML Web pages. As I have mentioned, the real challenge is creating DHTML so that you reach the largest possible audience within certain practical limitations. I say "practical" because true DHTML is not possible in any Web browser that doesn't implement scripting or style sheets, and impractical in those that suffer from partial or poor implementations. In other words, using DHTML, you'll never be able to reach 100 percent of all possible people.

The good news is that with a firm understanding of the differences between the leading Web browsers and some creative coding, you can create cross-browser DHTML Web pages that will perform some, or most, of the things you want, using one Web page.

The alternative is to create multiple versions of your Web page, targeted at the different "sects" of the Web browser market. I don't consider this a practical or tidy solution because you will end up multiplying your workload by two, three, four, or more times. Therefore, I will continue to focus on cross-browser DHTML in the sections that follow.

Technology: Enterprise Software

by Danny Bradbury

Thursday 16 November 2000

Scripting languages

Web addresses no longer end with ".html" - a host of scripting languages are available to jazz up your pages. Danny Bradbury offers a guide to .php, .asp and the rest

Gone are the days when producing a Web page simply involved writing some HTML code or painting a screen using Microsoft's Frontpage Web design tool. These days, with the Internet going into e-commerce overdrive everyone wants dynamic Web experiences. Scripting has taken a quantum leap.

There are two main categories of scripting language - either client or server-based. They are designed to describe attributes and functions that can be interpreted by browsers to produce a Web page.

Client-side Web scripting started off with hypertext markup language (HTML), which was a static scripting language used purely to describe how a page would look. You can use HTML, for example, to position a headline, and decide which colour it will be.

HTML has come a long way since it was first developed as the basis for the World Wide Web at the start of the 1990s. The World Wide Web Consortium (W3C), the industry body that ratifies some Internet standards, has released version 4.0 of the technology.

In the late 1990s HTML spawned dynamic HTML (DHTML). This language enables the browser to interact with the end-user, creating pop-up windows and so forth. But DHTML had to compete with some other client-side scripting languages, in the form of VBScript and JavaScript.

VBScript was designed by Microsoft and is a scripting version of its Visual Basic programming language. The problem with the technology is that although it offers great functionality (it is also used in Microsoft Office to customise applications) it is only understood by Microsoft Internet Explorer.

Much more popular these days is JavaScript, which was released as part of Netscape's Navigator browser in September 1995. Microsoft quickly followed with its own implementation, and unfortunately the two were not initially compatible. The European Computer Manufacturers Association (ECMA) developed a standard version in June 1997, and the browsers are now much more in line with one another.

JavaScript can be used to do multiple things, such as validating forms on the client, creating animations and changing your experience of a site according to the time of day, for example.

The row over browsers, however, combined with the proliferation of different client devices such as WAP phones, has led to a slow departure from client-side scripting in favour of server-side scripting. Processing everything on the server means that you can give everyone a similar experience of your Web site, while making allowances for different display types.

One of the first scripting interfaces for the server was the common gateway interface (CGI), which enables applications to interpret scripting languages, carrying

out different functions as a result. Perl is one of the most common languages used to write to CGI, although this language is hardly intuitive to use.

Microsoft developed active server pages (.asp) as a means of taking inputs from a Web page (from a form, for example) and processing them so that they can interact with objects on the server. This means the input could be used to look up a database, for example. Once the processing has been completed the active server page can then take the output and render it into HTML for display in the browser.

Sun Microsystems responded with Java server pages (JSP) another scripting language that differs because the scripts are compiled and loaded as servlets - small programs sitting on the **Web server**. Compiled programs are generally faster than interpreted ones, so JSP applications can provide performance advantages (see box above).

According to documentation from **software development** company Rational, most of an application's business logic should not be held in a scripted page. Rather, it should be held in the business objects that the page interacts with. The server-side scripted page should essentially be the way for the browser to talk to a server-based program.

One of the biggest steps when moving from a static environment to a server-side scripted environment is knowing how the scripts will interact with the middle tier, which contains all of the complicated programming logic that drives the application.

This means that you must have a thorough understanding of the technical architecture of the application, and it also means that if the application changes, the scripting must be regression tested - tested with the new code - to make sure that it still works properly.

One advantage to server-side and client-side scripts is that they are easy to implement. Rather than having to learn a complicated language like C++ or Java you can pick up much scripting functionality in the course of a few days.

I learned how to program many features in JavaScript by reading a book that only took me two days. It is an excellent way to take advantage of your Web-based application quickly and easily.

But don't let the ease of implementation tempt you into undisciplined development. You still need to observe conventional procedures and safety measures when changing your code.

Next week: Danny Bradbury looks at browser wars and their aftermath.

Learn your lines - a guide to Web scripts

- Javascript: Client-side scripting language, developed in 1995.

- REXX: A scripting language for IBM mainframes, developed in the late 1970s. Later versions developed for DOS and Windows.

- TCL: Tool Command Language, used for processing strings and passing commands to interactive programs.

- ASP: Active server pages. Developed by Microsoft for use

with its Internet information server [IIS]. Scripted pages sit on the Web server and provide an interpreted interface between the browser and the back-end application.

- JSP: Java server pages. Sun Microsystems' answer to active server pages. These scripted pages are compiled and run on the server as small programs called servlets. They do much the same thing as active server pages.

- PHP: This open source server-side scripting language is embedded into HTML, much like other scripting languages, and provides an interface between the browser and the back-end program.

Print this page >>

Send to a friend >>

Subscribe to E-mail >>

A-to-Z index >>

the complete webmaster

tutorials reviews reference

ASP
CGI
FrontPage
HTML
Java
JavaScript

home / articles / javascript

Browser targeted Cascading Style Sheets using JavaScript

Cascading Style Sheets (CSS) received a great deal of attention during 1998, so if you are involved in the creation of websites there is a good possibility that you experimented with the use of them on your site.

Unfortunately, the currently available range of web browsers have inconsistent support for CSS. As a result, what looks good in one browser may not always look good in another. Webreview.com has a good summary of the differences between CSS support in the major browsers.

In fact, due to the major differences between the two most popular web browsers in common use (Internet Explorer 4 and Netscape 4), experienced web developers may use different CSS files for each browser, or consider splitting their site into a Netscape enhanced section, and an Internet Explorer enhanced section. One way of doing this is to use browser detection on the server [for an example, see The Complete Webmaster's article on server-side browser detection using ASP].

Unfortunately, server-side browser detection can add significantly to server load. It can also prevent the effective caching of pages, contributing to an increased Internet bandwidth usage. Finally, not all web developers will have access to the facilities for detecting browsers on the server. An alternative is therefore to use JavaScript. This scripting language is understood by both Netscape and Internet Explorer. Furthermore, because it is executed by the web browser it does not require any additional server resources.

A simple browser-detecting JavaScript

Writing a JavaScript to detect whether the page is being viewed in either Netscape or Internet Explorer is not difficult. For example, the script below will inform you of whether you are using Internet Explorer or Netscape:

You appear to be using Microsoft Internet Explorer

Because CSS are only supported in versions of Netscape 4 and IE 4 or

above (the limited CSS support in Internet Explorer 3 is not worth bothering with), the script will only detect version 4 (or greater) of both browsers. The script required for this example is below:

```
<script language="JavaScript"><!--
browser_version= parseInt(navigator.appVersion);
browser_type = navigator.appName;

if (browser_type == "Microsoft Internet Explorer" && (browser_version >= 4)) {
document.write("<I>You appear to be using Microsoft Internet Explorer</I>");
}
else if (browser_type == "Netscape" && (browser_version >= 4)) {
document.write("<I>You appear to be using Netscape</I>");
}
// --></script>
```

Adapting the JavaScript to load browser specific style sheets

Adapting the script shown above to load a CSS is not difficult. The script has to write a line of HTML to the browser, such as the HTML below:

```
<link REL="stylesheet" HREF="../_private/MyStyleSheet.css" TYPE="text/css">
```

This HTML should appear in the HEAD portion of the HTML document.

The script to load a specific CSS according to the browser being used is shown below:

```
<script language="JavaScript"><!--
browser_version= parseInt(navigator.appVersion);
browser_type = navigator.appName;

if (browser_type == "Microsoft Internet Explorer" && (browser_version >= 4)) {
document.write("<link REL='stylesheet' HREF='012899-ie4.css' TYPE='text/css'>");
}

else if (browser_type == "Netscape" && (browser_version >= 4)) {
document.write("<link REL='stylesheet' HREF='012899-netscape4.css'
TYPE='text/css'>");
}

// --></script>
```

Don't forget that this script has to be placed in the HEAD part of the HTML document. Some older HTML editors (e.g. FrontPage 97) don't like scripts placed in the HEAD part of the document, so be aware of this.

An example page demonstrates the script. If you view the page in Netscape 4 (or above) then the style sheet will cause the browser to load this style sheet, which specifies that the headings and background image should all be in an orange color scheme. Alternatively, if the page is viewed in Internet Explorer 4 (or above) then an alternative style sheet is loaded which specifies that the headings and background image should all be in a blue color scheme.

If you don't have both of these web browsers, then these screenshots will show the appearance of the page in both Netscape 4.5 and Internet Explorer 5 (beta).

Further considerations

Don't forget that the World Wide Web is viewed by a massive audience using a wide range of hardware and software. Not all web browsers support style sheets. Furthermore, Internet Explorer 4 and 5 allow users to override style sheets with their own. Before using this script, you might therefore like to check to see what type of browsers users are visiting your site. If you don't have access to your server's logfiles, then try using a free web statistics service like ShowStat, or one of the many other counter sites listed at CounterGuide.Com. Many of these will show you which browsers your site visitors are using.

Author: Brett Burridge
Date: 01/28/1999

More articles about JavaScript
More articles by Brett Burridge
Author Biography

An Improved Method for Creating Dynamic Web Forms Using APL

Steven J. Halasz
Lingo Allegro U.S.A., Inc.
Web: http://www.lingo.com
Email: sjh@sjhalasz.com

Abstract

The subject of this paper is "active" or "dynamic" web forms handled by Dyalog APL running on a web server. With "active" web forms, the HTML which specifies form controls is generated dynamically by a server program, in contrast to "static" web forms in which the HTML is fixed and does not vary from one page hit to the next. Also, I will be discussing only "thin client" approaches which minimize the amount of code which runs on the client machine. The method shown, which I call a "standard template" method, allows source pages to be maintained with visual tools such as Macromedia Dreamweaver and minimizes the amount of coding required when new forms are created in the same general format.

Thin and Dynamic

With active web forms, the HTML sent to the client browser calling a particular page may be different for different clients at different times. For example, the HTML specifying the choices available for a combo control or a radio control may be generated by the server program depending on the web user's particular options or circumstances.

"Thin client" approaches minimize the amount of code which runs on the client machine, which provides the benefit that the web page loads faster and is more easily made compatible across different browsers. A "fat client" approach would require Java, APL, or other programs to be downloaded to the client in order for the page to load.

Why be concerned about thin client versus fat client? To the extent that programs are downloaded to the client, the advantages of the web browser model of computing are lost. By using the web server model rather than the client-installed client-server model, you may avoid problems with hardware and software incompatibilities on the client machine. The following are the variables that may affect performance of programs on the client machine:

- operating system version,
- browser version,
- hard disk storage available,
- memory available,
- monitor size and resolution,
- chip speed,
- memory speed,
- hard disk speed,
- other applications running.

The less you rely on the client machine's configuration and loading, the less you are likely to stumble over these issues. In the most extreme case, a "fat client" web page may be little different from a client-server program installed on the client machine, except that the program installs itself automatically.

Conventional Methods

In general, it's not difficult to create active web forms, of course. There are a number of choices. You can use a program written in Perl or APL for example to generate the complete HTML page returned to the client browser. In that case, however, you need to maintain all of the HTML for the page by modifying the program, even though only a small part of the HTML may vary from one program call to the next.

A better approach is to create a "template" HTML page--a file which is modified by a server program before being sent to the client browser. That way, the template HTML can be changed by someone who is

APL00, 07/00, Berlin, Germany

This is the approach used by the very popular "Cold Fusion" HTML pre-processor and by the Lingo Allegro ISAP product for Dyalog APL. Special non-standard tags are added to the HTML in the template document which are pre-processed by Cold Fusion or APL. Let's call this the "non-standard template" approach, because the template contains tags which are not standard HTML.

The problem with the non-standard template method is that the source document cannot conveniently be developed and maintained using "visual" editing tools such as Microsoft Front Page (which, however, makes a terrible mess of the HTML such that it usually doesn't work with Netscape) or Macromedia Dreamweaver. These tools don't know what to do with the non-standard Cold Fusion or ISAP tags, and so the advantages of visual editing of the web page are lost. You will probably use a text editor to create non-standard template HTML files rather than a visual tool.

Standard Template Method

The approach I am presenting I will call a "standard template" method. The HTML template contains only standard HTML tags. Even so, it can be pre-processed by APL on the server to dynamically modify HTML controls on the page as required.

I have used this "standard template" approach to create a web page that is analogous to a typical client-server GUI form installed on the client PC (see Figure 1 below). Such a form has

- a menu bar at the top to navigate to other parts of the application
- a tab bar to navigate to different sections of the form
- the form itself, which contains the controls

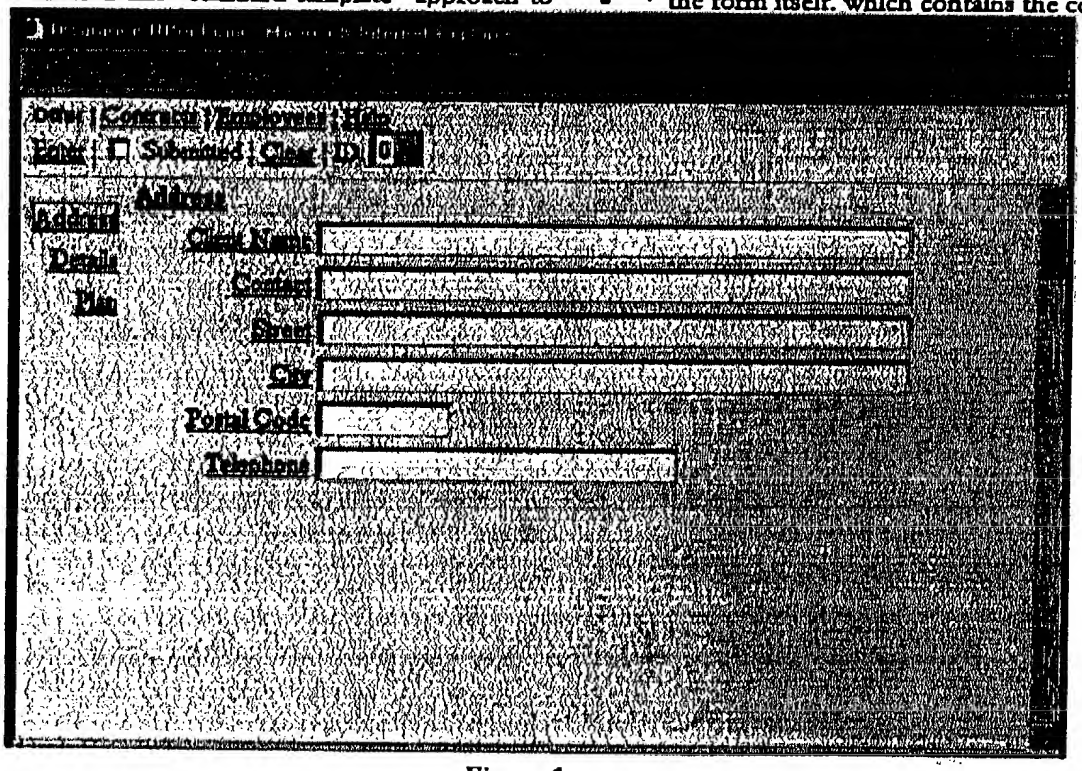


Figure 1

This GUI form layout is translated to a web page with three frames which correspond to the these elements of the GUI form. The top frame contains a menu bar and other controls which apply to the form as a whole. The left frame is like a tab bar which navigates to different sections of a form. The middle frame contains the form itself.

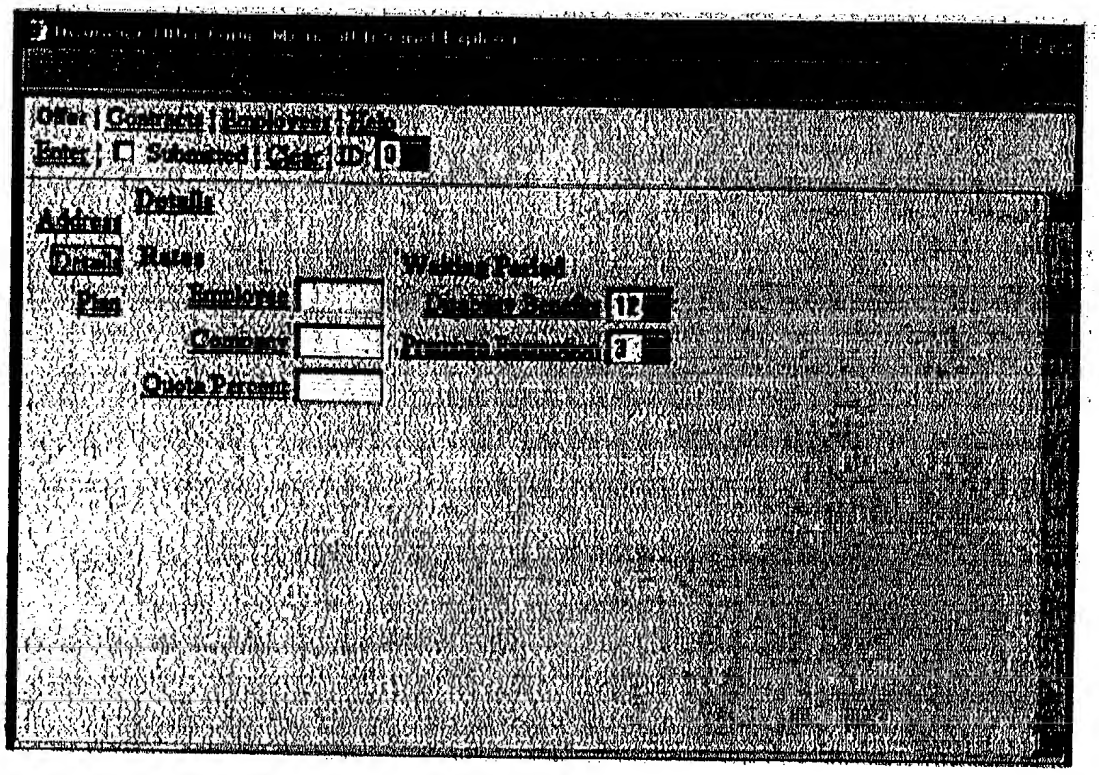


Figure 2

The example form accepts life insurance applications from insurance agents. The insurance policies cover employees of the agent's client

Document Oriented Interaction

The form uses a model of interaction which is a little unusual, and which I will call "document oriented", because it is based on interaction with the system by creating and submitting documents, something like the models employed by Lotus Notes and SAP.

Each insurance agent may submit multiple applications, one for each of several insurance customers. Each such application is an "instance" of the application form.

When an insurance agent uses the form, he is identified by his log-on. Windows NT Server allows you to create user IDs authorized to access pages from a particular Windows directory, and will challenge a user to log on with user name and password when accessing pages from that directory. The user ID is reliably reported to the Dyalog APL program in the USER environment variable.

For each insurance agent who logs on in this way, a separate collection of the agent's "instances" of the form is maintained in a temporary table. This temporary table, by convention, has the same name as the corresponding HTML file. The agent can add, change, or delete instances of the form maintained in the temporary table.

When the user is satisfied with the information provided on the form and all input validations are satisfied, the user clicks the "Submit" button and only then is the form submitted for processing. The user can still view the form but can no longer delete it or make changes. A separate process is triggered off-line to perform whatever processing is required when such forms are submitted.

This model provides several important benefits:

- The user does not lose input between sessions if the form is incomplete or does not pass input validations.
- The user can start a new form for a new client even though the form entered for a previous client is incomplete.
- Enhancing and debugging the form is simpler when the user interaction is isolated from processing of the submitted form.

- Off-loading the processing of the submitted form can improve response time of the web server and may also provide security benefits.
- It allows an exact one-to-one correspondence to be maintained between form controls and fields in the temporary table, simplifying development of the form interaction.
- Web form entry can remain active even if the main enterprise database is unavailable for any reason

.

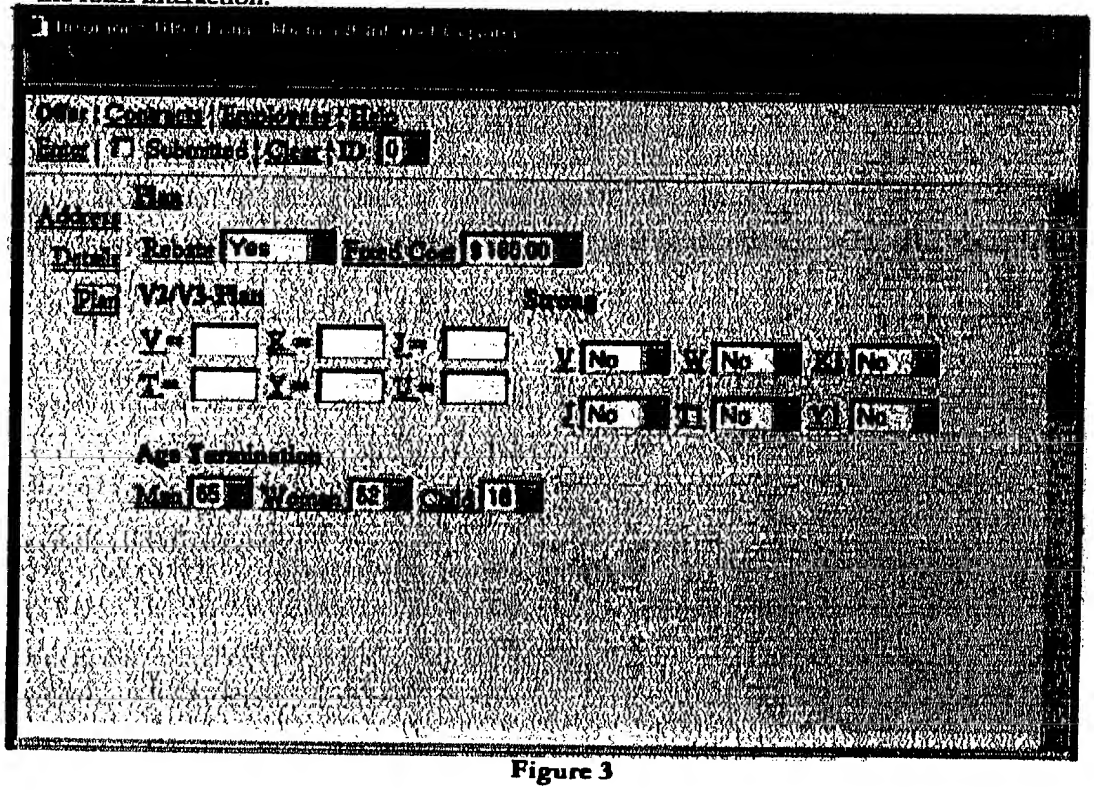


Figure 3

The Sample Form

Let's look first at the middle frame which contains the form itself. It is in three sections. The first section contains edit controls to accept the employer's identifying information. The second section contains details about the offer (see Figure 2). The third section specifies the particular plan selected (see Figure 3). You can scroll to different sections of the form using the scroll bar. The form controls must be laid out in cells of an HTML table, in order to control the layout of the form elements when controls change dynamically.

The left frame contains links to anchors in the main form. Clicking on one of these links causes the form to scroll to the anchor.

The top frame contains navigation controls which remain fixed at the top when scrolling to different parts of the form. The top line contains menu choices to switch to different parts of the application. The second line contains the following controls:

Enter Submits form data to the server (like the enter key on a mainframe terminal). Form data is validated and stored temporarily on the server, but the form is not submitted.

Submit Submits the form for processing.

Cancel Clears the form, and deletes the form instance if it is not submitted.

ID A combo control for the user to select an instance of the form belonging to the particular user.

How It's Done

So, how is this implemented? Lingo Allegro sells software products which allow you to configure a web server so that client requests for files having a specific extension, for example, DLG, are preprocessed by Dyalog APL before they are passed on to the client browser. Our ISAP product, developed by Andrei Kondrashev, is for use with Microsoft Internet Information Server, and our CGIAP product, developed by Dmitri Zlobin, is for use with any web server that supports standard CGI web programming.

The Dyalog workspace receives CGI variables and environment variables from the server. The CGI variables contain form input values from the client browser. The environment variables contain the user ID of the user and the name of the HTML file requested.

The program reads the HTML file requested and parses the tags, creating an APL array which represents the HTML tags in the file. By convention, all tags which are to be processed by APL are required to be named, that is, they are required to have a NAME= or ID= attribute. The APL program references and modifies the tags in this APL array by name. Also by convention, tags for controls that accept input must have the same name as the corresponding field in the temporary table. The array is stored in a temporary file and is only updated when the source document changes.

What does this accomplish? With this method, it's not necessary to have any non-standard tags in the HTML. In ISAP and Cold Fusion, by contrast, it's necessary to have non-standard tags which identify the database and map form controls to database fields. Adding an intermediate temporary table which is dedicated to handling the form separates user interaction from processing the submitted form, and greatly simplifies the creation and maintenance of the web form.

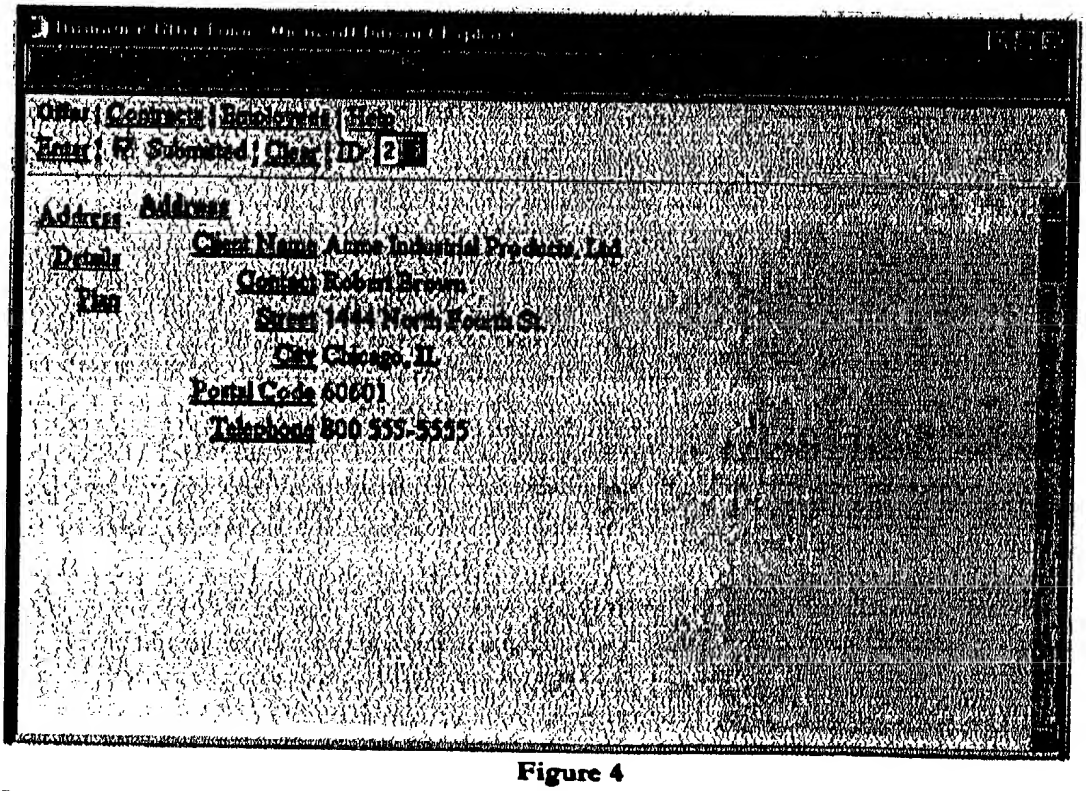


Figure 4

The top frame and the middle frame are both handled by APL. For the top frame, APL sets the list of available form instance numbers in the ID combo control. For the middle frame, APL validates user input and stores the user's input in the temporary table. When the user navigates to a different instance of the form using the ID combo, APL sets the controls according to the values in the temporary

table. When the form has been successfully submitted, APL returns the form contents as static text rather than as controls (see Figure 4).

A Clever Trick

I have used what I believe is a slightly clever way of handling input validations with this method. Normally, HTML has no way of associating a control such as an edit control with the label for that control. I am using a non-standard attribute, LABEL=, which specifies the name of the label associated with that control. The label is defined as a hyperlink rather than as text. Clicking on the label for a field navigates to a field-specific help page which is "active", that is, has the extension DLG and is pre-processed by Dyalog (see Figure 5). When a user input does not pass input validation, the attribute COLOR=RED is added to the hyperlink tag for the label, and clicking on the hyperlink in that case will give the field-specific help message enhanced with specific information as to how and why the user's input fails validation.

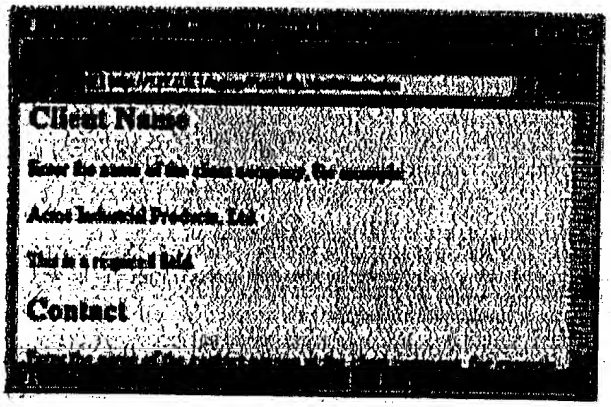


Figure 5

Using this method, off-line processing of the submitted form may perform additional validations which require accessing the main database, and which may require human review and approval. If a form fails these additional validations, the form submission task would "un-submit" the form and write an appropriate message into the temporary table for that record.

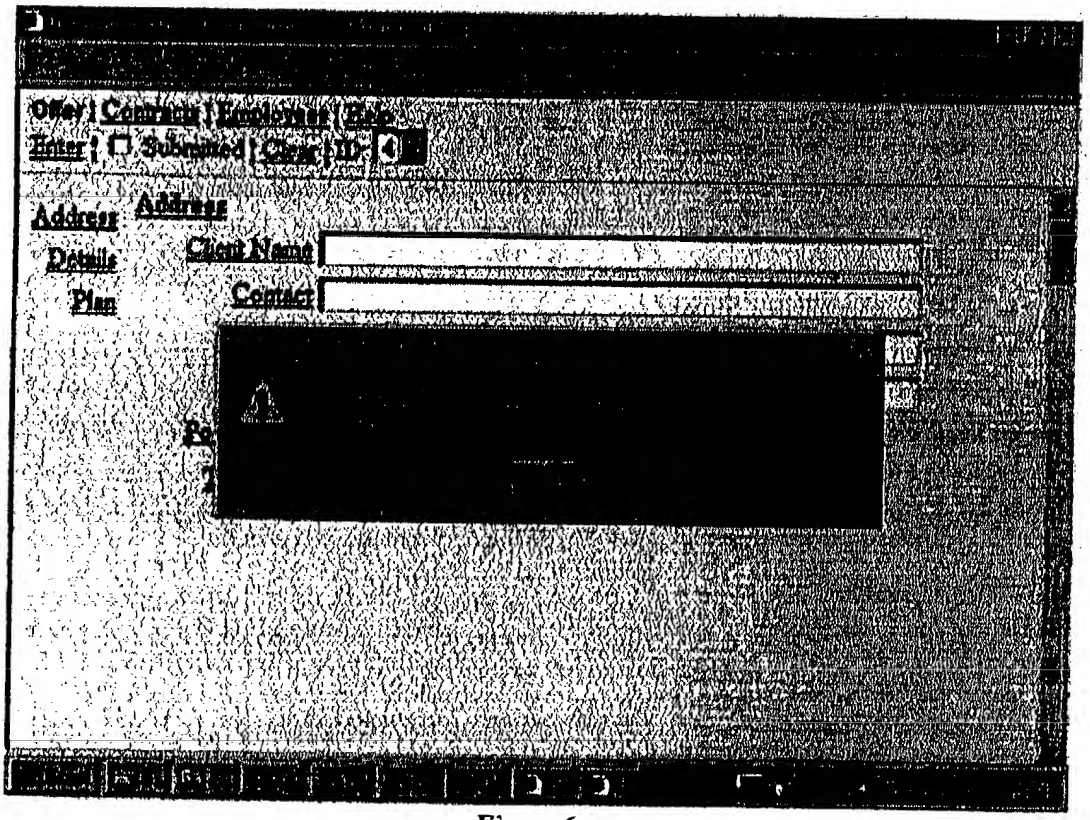


Figure 6

A Generalized Tool

An important benefit of this method is that almost all of the Dyalog APL programming is generalized and will handle any form that can be laid out in this format. A non-standard attribute such as "VALID=F7.2" can be used in the control's HTML to specify ordinary numeric and date validations of edit fields. Only the more complex validations require custom programming in APL. To separate the generalized APL from form-specific validations, the form-specific APL is kept in a separate workspace which is copied in using]CY on demand. This code will normally be very small and will load quickly because Dyalog's]CY is very fast. This allows the same generic code to be used for all similar forms.

So, to add a new form based on this model, it is only necessary to

1. create HTML for the middle frame using, for example, Dreamweaver, making sure that each form control has a name attribute and using the non-standard attributes LABEL= and VALID= as required
2. create a temporary table having the same name as the form and having fields with the same name and appropriate data type for corresponding form controls
3. write any required custom input validations for the form
4. write an off-line program which reads a form instance from the temporary table and processes it as a submitted form.

Some JavaScript Required

This method requires that the web page use a small amount of JavaScript. However, the JavaScript used is generalized and does not need to be modified for each different form which follows the specified format.

For the top frame, JavaScript performs the following operations

- when the frame is loaded:

1. The ID number of the current form instance is read from a hidden variable in the middle frame and the ID control is set to display that ID.
2. The SUBMITTED hidden variable is read from a hidden variable in the middle frame and used to set the SUBMITTED check box.

- when the user selects a navigation option:

1. the name of the button selected is written to a hidden variable in the middle frame
2. the form in the middle frame is submitted

- when the user checks the SUBMITTED checkbox:

1. the state of the SUBMITTED hidden variable in the middle frame is set
2. the form in the middle frame is submitted

- when the user selects a different form instance from the ID combo control:

1. the ID number selected is written to the hidden variable ID in the middle frame
2. the form in the middle frame is submitted

For the middle frame, the following operations are performed by JavaScript:

- when the middle frame is loaded:

1. the top frame is reloaded if necessary to update the list of form instance IDs in the ID combo control
2. if the hidden variable MESSAGE is not empty, a pop-up message box is displayed giving the message (see Figure 6 above).

Security Issues

Security is handled by the web server software and is of no concern to APL. Users log on to the operating system using standardized user identification and validation, and are identified in APL by their user name present in the USER environment variable.

Conclusions

You can try out and work with the demo yourself at http://sjhalasz.com/cgiap/offer.htm.

As usual, with APL it is possible to create a tool which goes well beyond what is generally available in terms of power and ease of use. The method I have described falls in the category of "special purpose software tools", an arena for which APL is very well suited. Normally, only very general purpose tools are developed using C because of the cost and burden of developing the tools. With APL, it's practical to develop tool kits or development environments for very specialized applications. An APL application developer can create and enhance such specialized software tools as required for an specific application. In contrast, an application developer using another technology usually cannot, as a practical matter, develop sophisticated special purpose tools, but must rely only on very generalized tools developed and

maintained by others. This aspect of APL is one reason why APL still has the power to astonish customers with its power to rapidly deploy sophisticated applications.

Steven J. Halasz

Generating dynamic content at database-backed web servers: cgi-bin vs mod_perl

Alexandros Labrinidis
labrinid@cs.umd.edu

Nick Roussopoulos[†]
nick@cs.umd.edu

Department of Computer Science and Institute for Systems Research,
University of Maryland, College Park, MD 20742

Abstract

Web servers are increasingly being used to deliver dynamic content rather than static HTML pages. In order to generate web pages dynamically, servers need to execute a script, which typically connects to a DBMS. Although CGI was the first approach at server side scripting, it has significant performance shortcomings. Currently, there are many alternative server side scripting architectures which offer better performance than CGI. In this paper, we report our experiences using *mod_perl*, an Apache Server module, which can improve the performance of CGI scripts by at least an order of magnitude. Except for presenting results from our experiments, we also briefly describe the implementation of an industrial strength database-backed web site that we recently built and give a quick overview of the various server-side scripting mechanisms.

1 Introduction

In less than 10 years from its debut in the early 90s, the Web has changed our lives dramatically. From comparing prices and shopping online, to viewing realtime stock quotes and managing our bank accounts, the Web is increasingly being used as the means to do everyday tasks. One common denominator for all these activities is the need to generate *dynamic content* [B+98]. Personalization, frequent updates, and searching/querying capabilities are the most common reasons behind dynamically generated web pages.

In order to generate dynamic content, web servers need to execute a program, through some *server-side scripting mechanism*. This script typically connects to a DBMS, performs a query, retrieves the results, and formats them in HTML in order to be returned to the user. Although cgi-bin was the first approach at running programs at the web server, it was not designed to handle the demand for dynamic content creation that web servers face today.

A plethora of server-side scripting mechanisms have been proposed to replace cgi-bin [Gre99, Mal98, FLM98]). One popular mechanism is *mod_perl*, which is a module that can be used with the Apache Server to support efficient CGI-like server-side scripting.

Instead of having a program that generates HTML, the various forms of *annotated HTML* embed scripting commands within an HTML document. The web server parses and executes these commands, before the final HTML page is sent to the user. The most popular annotated HTML approaches are three. 1) PHP [PHP] is free, open source software with many features and support for practically all DBMS platforms. PHP also supports persistent database connections. 2) Active Server Pages [ASP] is Microsoft's solution to server-side scripting and is supported by the Internet Information Server, Microsoft's Web server. ASP scripts can connect to SQL Server, Access, Oracle, Informix or any ODBC-compliant database. 3) JavaServer Pages [JSP] is SUN's ap-

[†]Also with the Institute for Advanced Computer Studies, University of Maryland.

proach to server-side scripting, that uses XML-like tags and scriptlets written in Java to encapsulate the logic that generates content for the page. JavaServer Pages can be used together with the Java Servlets architecture.

Java Servlets [SRV] are another approach at dynamic content generation. They are protocol and platform independent server-side components written in Java. Servlets dynamically extend Java enabled servers.

There is no simple answer when deciding which server-side scripting mechanism is best for a particular setting. However, if moving from or upgrading an existing CGI-based application, then the solution which dramatically improves performance and has the smallest *migration cost* is clearly mod_perl.

In the next section we present the various CGI-based server-side scripting mechanisms in more detail. In Section 3, we describe the implementation of a real-life application, and in Section 4 we present the results of performance experiments comparing cgi-bin to mod_perl. Finally, we discuss our conclusions in the last section.

2 Server-Side CGI Scripting

2.1 The CGI protocol

CGI[1] is a simple **protocol** that specifies the way in which user-defined scripts that run at the web server can communicate with users' browsers. Scripts that follow the CGI protocol are called *CGI scripts*. A CGI script operates in a rather straightforward way. First, it gets invoked by the web server, reads the user's input, which was typically submitted through an HTML form, and parses it. Then it does whatever processing is required and, finally, the script generates an HTML page that is returned to the user (Figure 1).

The CGI protocol dictates how form data should be passed from the web server to the script. Specifically, the data must be transformed into one long string of name-value pairs like: "name1=value1&name2=value2&....", after the values have been URL-encoded by converting spaces to + signs and special characters to %xx

[1]CGI is short for Common Gateway Interface.

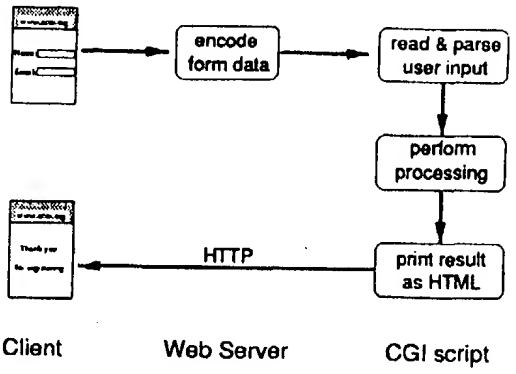


Figure 1: Execution of a CGI script

sequences, where xx is the ASCII value for the given character. For example, the = sign should be converted to %3d (see [Gun96] for more details). When a CGI script is activated, it must parse this long string into name-value pairs, doing the inverse transformations.

The CGI protocol also mandates that the standard output of the script is forwarded to the user. This means that the CGI script is expected to print the necessary HTTP headers first, followed by the results page in HTML.

Finally, it should be pointed out that the CGI protocol does not specify how the scripts are to be invoked by the server. Also, the CGI protocol does not restrict the choice of programming language for writing the scripts. The ability to use one's favorite programming language is one of the main reasons behind the popularity of CGI scripts.

2.2 The cgi-bin invocation mechanism

The cgi-bin invocation mechanism was adopted in the implementation of CGI scripts by the first web servers (CERN & NCSA). Each time the server needs to run a CGI script, it has to spawn a new process, setup the CGI environment, run the script, and send the script's standard output to the user's browser (Figure 2).

This approach to server side-scripting has some advantages. First of all, since it follows the CGI protocol, the choice of programming language is not restricted. The second big advantage to forking a new processes for every CGI script is that it

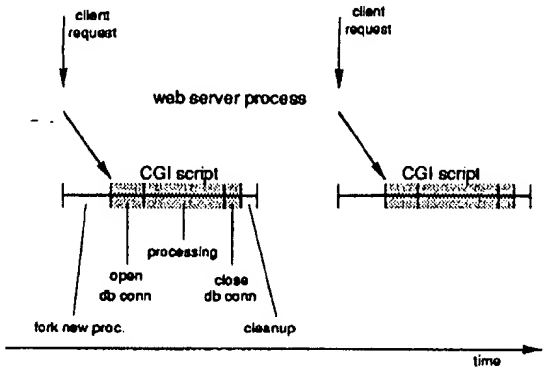


Figure 2: cgi-bin script invocation

provides an easy separation between the web server and application scripts; "bad" programs will not crash or slowdown the server.

There are however many disadvantages to using the cgi-bin invocation mechanism, the most important of which is the poor performance of the web server, especially under high loads. Forking a new process for every CGI script has a big operating system overhead. Moreover, each script is expected to have an initial setup phase, in order for example to establish a connection to the DBMS, initialize variables, etc. Since the lifespan of CGI scripts is typically very small, all this setup overhead is "wasted" and has to be repeated with every script. To overcome this performance problem, a number of solutions have been proposed like FastCGI [FCG], which attempts to eliminate the process spawning overhead and mod_perl, which incorporates the CGI scripts inside the Apache Server, as we see in more detail in the next section.

Finally, another disadvantage of the cgi-bin approach are the long and ugly URLs like *http://www.cs.umd.edu/cgi-bin/search.pl* that must be used.

2.3 The mod_perl invocation mechanism

Mod_perl [MP] is a server scripting module for the widely popular[2] Apache web server [AS]. CGI scripts on a mod_perl-enabled Apache Server run **within** the server processes (Figure 3), thus eliminating the overhead of spawning a new process with every client request. Furthermore, the scripts can

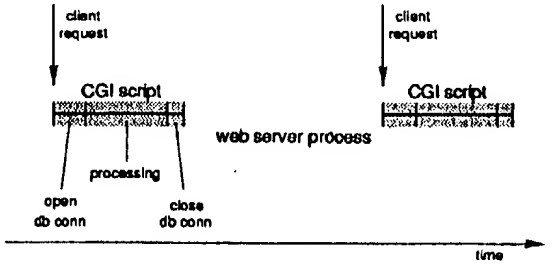


Figure 3: mod_perl script invocation

store state information across multiple invocations and can, for example, maintain open connections to the DBMS, thus avoiding the cost of establishing a connection with every request (Figure 4). Overall, the mod_perl approach minimizes the setup overhead for serving CGI scripts.

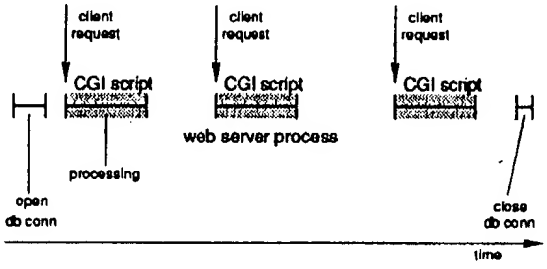


Figure 4: mod_perl script invocation (with persistent database connections)

The Apache Server is well designed and very robust, so that, although mod_perl scripts run within the server processes, "bad" scripts cannot crash the web server. For example, each script can service at most *MaxRequestsPerChild* requests, after which it must be reloaded, thus cleaning up any possible memory leaks it may have had.

Another advantage of the mod_perl approach is that CGI scripts have access to the Apache Server internals via an API, which increases their functionality. A direct consequence of this is being able to get rid of the ugly URLs that were required by the cgi-bin approach. For example we can easily have *http://www.cs.umd.edu/search* instead of *http://www.cs.umd.edu/cgi-bin/search.pl*.

[2] According to the December 1999 Netcraft Server Survey, an estimated 54.5% of web sites over the world use the Apache Server [NSS99].

One last advantage of mod_perl is the ease of migration from existing CGI scripts which were written to work with cgi-bin. Since most of these scripts were written in Perl or C, the transition is straightforward [CMP].

The mod_perl approach has only one disadvantage: the choice of programming language is somewhat restricted, since it has to be Perl or C. However, we do not believe this to be a real problem as these two languages are the most popular ones for writing CGI scripts.

3 A real-life application

In this section, we briefly present our experiences from implementing the *dbgrads system* [DBG], a real-life application on a database-backed web server. The dbgrads system is a searchable database of students that have a background in databases and will be graduating soon. It is intended to be used by prospective employers from academia or industry around the world in order to locate candidates for their job openings.

3.1 Software checklist

Before building the dbgrads system, we had to install a few software packages first: the web server, the DBMS and a few Perl modules. All of the software was free and publicly available.

Web server We used the Apache Server [AS], version 1.3.9, which, except for being free, is fast, efficient, portable, well supported, stable, reliable, extensible, easy to administer and has many features [SM99]. After having installed the Apache Server, we installed the mod_perl module [MPG] version 1.21.

DBMS We chose MySQL [MSQ, YRK99] version 3.22.27 as the database server, primarily because it is free, light-weight and can run as a normal user process (i.e. without super-user privileges) thus minimizing any possible security risks. In order for our scripts to communicate with the MySQL database server, we used the Perl DBI module, a generic Perl interface to relational DBMSs, combined with the DBD (Database Driver) module for MySQL (both available from CPAN [CPN]).

Perl Perl version 5.004 [PRL] was already installed in our system, as well as the CGI.pm module [CGP, Ste98], which is a very useful library for writing CGI scripts in Perl.

3.2 Implementation

Writing code for a database-backed web server is similar to any other code development project, with one notable difference being the need for rapid prototyping and short turn-around times. For that reason, although it is important to come up with an initial design document outlining the functionality of the entire system, most probably there will be a lot of revisions to it by the end of the project. Also, since the web application is to be released to the entire world, special care should be taken to guarantee security, thus preventing attacks to the system. Finally, care must also be taken so that the web application can handle all possible errors internally, without the user ever having to see the generic Error Message page that most web servers have.

One big advantage to using the Apache Server with the mod_perl module is that we can have *virtual URLs*. We can, for example, specify that the URL *http://www.acm.org/mysearch* will correspond to a dynamically generated document (as opposed to a static HTML page), that was computed by a special, user-defined, *content handler* Perl function [SM99].

One big advantage to using the Perl DBI/DBD modules, is that we are not tied to a particular DBMS vendor and can, almost effortlessly, switch to a different database server. For example, the `quote()` function, part of the DBI library, serves to this purpose by correctly quoting SQL strings.

Due to space limitation, we cannot present more information on the dbgrads system. However, for more details on the implementation of a database-backed web site, the reader is referred to [Gre99], [SM99], and [CT98].

4 Experiments

For our experiments, we used the Apache Server version 1.3.6 and, instead of MySQL, the Informix Dynamic Server with Universal Data Option ver. 9.14. Both the web server and the DBMS run on the same machine, a SUN UltraSparc-5 with 320 MB of main memory and a 3.6 GB Seagate Medalist disk, running Solaris 2.6. Multiple clients were generated through multi-threading on another SUN machine, which was on the same local area network to minimize network variations. Each client performed about 60 requests, one after the other.

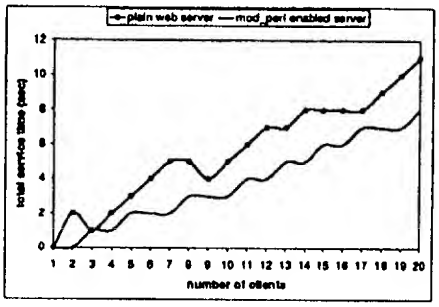


Figure 5: Delivery of static HTML pages

First of all we tried to see if there is any difference between the plain Apache web server and the mod_perl-enabled Apache Server, as far as delivery of static HTML pages is concerned. The results from our experiments are in Figure 5. The x-axis is the number of concurrent clients, and the y-axis is the time it takes for all clients to complete their request workload. We can see that the mod_perl-enabled Apache Server performs equally well with the plain server. In other words, the added functionality and the increased footprint do not affect the performance of static content delivery.

Figure 6 has the results from the second experiment, where we compared cgi-bin with mod_perl on dynamically generated web pages (that correspond to results from SQL queries). We studied two variations of mod_perl: plain mod_perl and *mod_perl+*, where the database connections are kept persistent. As expected, the mod_perl approach outperforms cgi-bin by at least an order of magnitude. Specifically, the performance of plain mod_perl is about 10 times better than that of cgi-bin, whereas the performance of mod_perl+ is about 20 times better than that of cgi-bin. This means that just by keeping the database connections persistent we cut the response times in half.

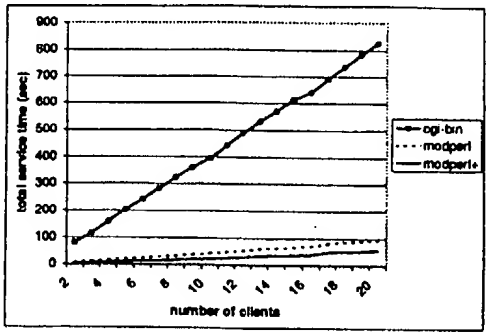


Figure 6: Comparison of cgi-bin to mod_perl

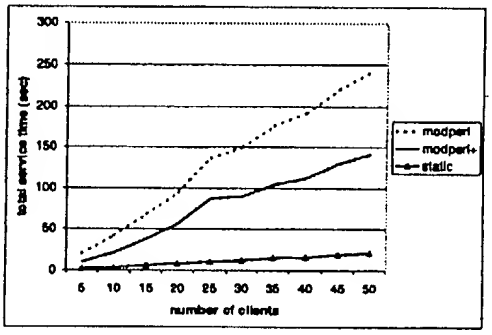


Figure 7: Scaling experiment

In our last experiment, we scaled up the number of concurrent clients and measured the performance of mod_perl, mod_perl+ (mod_perl with persistent db connections) and static (static HTML pages, served by a mod_perl-enabled Apache Server). Although the performance of static pages is, as expected, the most scalable solution, both mod_perl variants scale well, with the mod_perl+ approach giving consistently the best performance for dynamically generated web pages.

5 Conclusions

In this paper we presented a brief overview of the various server-side scripting mechanisms that are used to generate dynamic web content. We also discussed some details from the implementation of a real-life application on a database-backed web server. Finally, we presented experiments to support that the cgi-bin script invocation mechanism is at least an order of magnitude slower than mod_perl, and should thus be abandoned.

References

[AS] Apache Server home page. *http://www.apache.org/httpd.html.*

[ASP] Active Server Pages help page. *http://www.4guysfromrolla.com/.*

[B+98] Phil Bernstein et al. "The Asilomar Report on Database Research". *SIGMOD Record*, 27(4), December 1998.

[CGP] CGI.pm home page. *http://stein.cshl.org/WWW/software/CGI/cgi_docs.html.*

[CMP] Quick guide for moving from CGI to mod_perl. *http://perl.apache.org/dist/cgi_to_mod_perl.html.*

[CPN] Comprehensive Perl Archive Network. *http://www.cpan.org.*

[CT98] Tom Christiansen and Nathan Torkington. *"Perl Cookbook"*. O'Reilly & Associates, August 1998.

[DBG] dbgrads home page. *http://www.acm.org/sigmod/dbgrads.*

[FCG] FastCGI home page. *http://www.fastcgi.com.*

[FLM98] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. "Database Techniques for the World-Wide Web: A Survey". *SIGMOD Record*, 27(3):59–74, September 1998.

[Gre99] Philip Greenspun. *"Philip and Alex's Guide to Web Publishing"*. Morgan Kaufmann, June 1999. Available at *http://http://photo.net/wtr/thebook/.*

[Gun96] Shishir Gundavaram. *"CGI Programming on the World Wide Web"*. O'Reilly & Associates, First edition, March 1996. Out of print, available at *http://www.oreilly.com/openbook/cgi/.*

[JSP] JavaServer Pages. *http://java.sun.com/products/jsp.*

[Mal98] Susan Malaika. "Resistance is Futile: The Web Will Assimilate Your Database". *Data Engineering Bulletin*, 21(2):4–13, June 1998.

[MP] mod_perl home page. *http://perl.apache.org.*

[MPG] mod_perl development guide. *http://perl.apache.org/guide.*

[MSQ] MySQL home page. *http://www.mysql.com.*

[NSS99] Netcraft Server Survey, Dec. 1999. *http://www.netcraft.com/survey.*

[PHP] php home page. *http://www.php.net.*

[PRL] Perl home page. *http://www.perl.com.*

[SM99] Lincoln Stein and Doug MacEachern. *"Writing Apache Modules with Perl and C"*. O'Reilly & Associates, April 1999.

[SRV] Java Servlets API. *http://java.sun.com/products/servlet.*

[Ste98] Lincoln Stein. *"The Official Guide to CGI.pm"*. John Willey & Sons, April 1998.

[YRK99] Randy Jay Yarger, George Reese, and Tim King. *"MySQL and mSQL"*. O'Reilly & Associates, August 1999.

Dynamic HTML: The Next Generation of User Interface Design Using HTML

Published: February 1, 1997

Microsoft Corporation

Introduction to Dynamic HTML

Dynamic HTML adds richer, more engaging user interfaces to the HTML presentation language, while also greatly reducing the workload of networks and servers. The object model provided by Dynamic HTML gives Web developers the ability to dynamically update the content, style and structure of Web-based content, while providing them with detailed control over the appearance, interactivity and multimedia elements required for a polished and exciting application. Developed in collaboration with the World Wide Web Consortium (W3C), Dynamic HTML is a feature of Microsoft® Internet Explorer version 4.0.

Hypertext Markup Language Today

Hypertext markup language (HTML), the underlying language for creating Web pages, is a standard for delivering document-oriented content across the Internet. It is used by browsers and authoring tools on numerous operating systems.

But as a tool for delivering rich, interactive applications, HTML is limited. Even the most interactive and polished Web pages look dull, tame and functionally challenged compared to any of today's top education or entertainment CD-ROM titles. Users notice the difference immediately. HTML does not allow developers to create equally appealing and interactive applications because it is essentially static. Any interactive feature of a page that, when activated, results in a significant change to the page's appearance requires new HTML code from the server, that is, another round trip over the Internet. This repetitive server contact makes pages less responsive and taxes Web server resources.

HTML's inability to accommodate large, dynamic changes to a page's appearance limits it as a tool for creating front ends to business applications. A task as seemingly simple as creating a master detail order form is very difficult with static HTML. Such tasks are dramatically simpler with today's client/server application development tools such as Delphi, Powerbuilder and the Visual Basic® programming system.

HTML has another limitation: While it can express textual content, it can't express multimedia capabilities such as positional control (moving objects in two dimensions), sprite effects, built-in visual transitions, audio mixing or inclusion of real-time interactive multimedia.

Dynamic HTML

Dynamic HTML extends HTML with an object model allowing scripts or programs to change styles and attributes of page elements (or objects) , or even to replace existing elements (or objects) with new ones. Other additions include multimedia and database features. Developed by Microsoft Corp. in collaboration with the W3C, Dynamic HTML adds the interactivity, database manipulation and extensibility needed for creating business applications, as well as the snap and polish needed for consumer-oriented applications.

Dynamic HTML allows developers and Web page designers to bring more creativity, control and sophistication to their Web sites. It increases the usefulness, attractiveness and enjoyment of the Web within current bandwidth limits, without requiring developers to abandon their current tools. Developers can use the HTML editors they use today and control the dynamic behavior of pages through the languages that they use today, such as JavaScript, Java™ and Visual Basic, Scripting Edition.

Developers have other options for adding dynamic behavior to their pages (such as writing custom embedded objects in Java, Visual Basic or C++). But these isolated regions of the page cannot take advantage of the rich layout, open formats and easy editing provided by Dynamic HTML. With Dynamic HTML, interactive behavior is fully integrated with the expressive power of HTML, and integrated directly into the browser's page display. If custom embedded objects are necessary, they can be simpler because they leverage the power of Dynamic HTML. Many tasks that formerly required custom embedded objects can now be done with scripts.

Dynamic HTML is fully compatible with the W3C Cascading Style Sheets (CSS) Specification, making it compatible with current browsers and existing HTML pages.

Examples of Dynamic HTML Use

Dynamic HTML can be used in browsers, business productivity applications, "edutainment" titles and more. Examples include the following:

Business applications. Dynamic HTML makes the Internet a more powerful tool for business use. Dynamic forms (e.g., master detail order entry, sales tracking and analysis, and employee benefits) can respond to user input, recalculate on the fly, and obtain additional information in the background. With these new capabilities, Dynamic HTML becomes a viable application development language for creating client/server front ends for business applications.

Interactive documents. While the hyperlinking built into the HTML model aids user navigation of Web documents, documents delivered over the Internet and intranets are essentially confined to a page-by-page design metaphor. Dynamic HTML changes that by making it possible to create a more interactive document that responds instantly to user actions. Following are some examples of how interactive documents can be built using Dynamic HTML:

- **Dynamic expansion.** When users conduct a typical Internet search, they receive a summary page that lists target Web sites. Obtaining additional information requires clicking on a listing and going back over the Internet in search of the Web page. With Dynamic HTML, search results pages can be programmed with scripts that provide a detailed synopsis of any listing when the mouse is passed over it, eliminating redundant fetches from the server.

- **Text effects.** Hyperlinks or other text elements can change style based on mouse or keyboard actions. For example, to get the user to click a specific

hyperlink, the designer could cause its font to grow (and an audio theme to grow louder) as the mouse pointer moves closer.

- **Table manipulation.** Tabular data such as price lists and search results can be sorted, filtered and viewed using the built-in local database engine. This provides a more "live" experience of the document than conventional HTML.

Entertainment and education. Interactive entertainment and education Web sites can include animated characters that respond to user input by moving anywhere in a 2-D plane; they can also, through z-positioning and scalable graphics, appear to move in 3-D space. Audio, such as music or voice-overs, can fade in and out to correspond with the characters' movements.

Benefits of Dynamic HTML

Dynamic HTML provides the following benefits:

More creative options using objects. Developers have more options for programming their pages creatively. The entire contents of the Web page are exposed as a collection of open, extensible, scriptable objects, regardless of the language used to program them. Dynamic HTML can capture and respond immediately to a user's actions. Web page designers can make the page act as they need, with fewer limits imposed by HTML.

Rich multimedia and layout. Web site designers can use rich effects such as moving sprites, animated washes of color and texture across text fonts, dynamic multichannel audio mixing, font and screen transition effects (e.g., swipes and fades), vector graphics for scalable, low-bandwidth images, and x-, y- and z-order positioning. This last capability allows objects to move in a two-dimensional plane, as well as in front of or behind other objects ("2.5-D"), without going back across the Internet to the server for instructions.

Lower server load. Using Dynamic HTML, developers have the choice of creating dynamic content on the client or on the server, to optimize for the best user experience. When processing occurs on the client, no round trips need to occur, eliminating additional network traffic, latency and server load.

More snap. Users can interact with a Web page as though it were an application, without having to communicate with the Web server for each specific user interaction. Dynamic HTML content can modify itself on the fly in response to user actions, dynamically altering the appearance or content of the Web page. Data manipulation can occur locally, not on the server, resulting in less waiting for users.

Built-in database support. Using built-in data binding, Web designers can provide pages that organize data on the fly, interactively, on the client system and without requiring a round trip to the server. For example, a user can dynamically sort a list of stock quotations by price or by price/earnings ratio, without requiring complex Java programming or abandoning the display richness of HTML.

Open, cross-platform support. Dynamic HTML will be included in Microsoft Internet Explorer 4.0 and packaged as a no-cost component for all platforms supported by Active Client, including Windows®-, Macintosh- and UNIX-based systems. In addition, vendors of other applications, browsers or tools can incorporate Dynamic HTML technology seamlessly and royalty-free into their products and even extend the functionality to meet their specific needs.

Dynamic HTML and Standards

Dynamic HTML is a standards-based technology, enabling easy use in conjunction with today's software, tools, browsers and other Internet components. Unlike other approaches, Dynamic HTML builds on and is completely consistent with the Cascading Style Sheets Specification recently ratified by the W3C consortium. Microsoft has submitted its object specification for HTML programming to the W3C for adoption as an open standard. Microsoft has been working closely with the W3C and will enhance future versions of Dynamic HTML and other Active Client technologies to conform to specifications recommended by the W3C.

Dynamic HTML pages can be extended by incorporating Java Applets or ActiveX™ Controls in Web pages, and can be scripted using VBScript, JavaScript or any other language that supports the ActiveX Scripting interface. Because of the open extensibility and the flexibility it provides, Microsoft is adopting Dynamic HTML across all of its tools and applications as its standard user-interface engine.

Availability and Third-Party Support

Dynamic HTML will be provided in Microsoft Internet Explorer 4.0 on 32-bit Windows platforms (the Windows NT® operating system and Windows 95), Macintosh, Windows 3.1 and key Unix platforms. Microsoft Internet Explorer 4.0 is scheduled for general release early in the second half of 1997 and is available to third parties for royalty-free inclusion in browsers, tools and client application software.

Appendix: A More Detailed Discussion of the Major Features of Dynamic HTML

Dynamic HTML includes several major features:

- Dynamic styles
- Positioning
- Dynamic content
- Filter, transition, and animation controls
- Data awareness

These features enable Web page designers to dynamically change the style and attributes of elements, as well as insert, delete or modify elements and their text even after a page has been loaded. Dynamic HTML automatically updates the display of the page to reflect these changes, including reformatting the document as necessary. Some of these features can be used without scripting, while others are exposed via an object model that can be accessed from scripts and other components within a page (e.g., ActiveX Controls and Java Applets). The object model is a superset of, and is therefore backward compatible with, the Javascript object model found in Netscape Navigator 3. Like Microsoft Internet Explorer versions 3.0 and 4.0, Dynamic HTML supports ActiveX scripting, which gives Web page designers the choice of scripting languages, including JScript™ development software (Microsoft's implementation of a JavaScript-compatible scripting engine) and VBScript.

These Dynamic HTML innovations work together holistically and synergistically — the whole is greater than the sum of its parts.

Dynamic Styles

Using Dynamic HTML, Web designers can dynamically change the style of every

HTML element in a document. In HTML, styles are specified as element attributes or via Cascading Style Sheets (CSS). The object model exposed by Dynamic HTML exposes every HTML element in the document, including its attributes and CSS properties. Using simple scripts, Web page designers can dynamically read and change the values of these attributes and CSS properties. For example, dynamic styles can be used to do the following :

- Show or hide an element. For example, the detail text of a bullet point could be hidden until the user moves the mouse pointer over the bullet.
- Change the size, color or other font properties of elements. For example, a title could be emphasized by enlarging the font and changing its color when the user moves the mouse over the title.
- Change the position of elements on a page (see Positioning below).

Unlike other browsers, Microsoft Internet Explorer 4.0 uses Dynamic HTML to dynamically change the style and content of a page at any time, even after it has been loaded. Dynamic HTML supports intelligent recalculation to rerender a page only if sections of that page change, including reformatting text paragraphs as needed. For example, if list items are shown or hidden, Dynamic HTML adjusts the other related items, including renumbering them where appropriate.

Positioning

Dynamic HTML supports x-, y- and z-order positioning of HTML elements, as specified in the W3C Working Draft on Positioning HTML with Cascading Style Sheets. This capability allows Web page designers to place elements such as images, controls or text precisely on the page. By placing objects in different z-planes, Web page designers can also cause the objects to overlap, specifying which element should be on top of which.

By manipulating object coordinates and other dynamic styles using scripts, Web page designers can move elements around a page, thus animating the page. Dynamic styles, positioning, transparent ActiveX Controls (supported in Microsoft Internet Explorer) and transparent images present Web page designers with a rich set of animation options.

This positioning capability has many uses in both consumer and business applications. Two-dimensional style layout functionality was first available from Microsoft using the HTML Layout Control, which shipped in Microsoft Internet Explorer 3.0. The syntax used by the Layout Control was previously submitted to the W3C. Since then, Microsoft has been actively working with the W3C to turn the syntax into a standards-based proposal. This positioning functionality represents the evolution of that 2-D functionality, the key difference being that Dynamic HTML uses native HTML and W3C-proposed syntax.

September 4, 1997 Editor's note: The HTML Layout Control technology, orginally released with Internet Explorer 3.0, is now natively supported by Internet Explorer 4.0. Please see the HTML Layout Control home page for further information.

Dynamic Content

In addition to changing the styles on a page, Web page designers can dynamically change the content of an HTML page. This capability can be used to insert or delete elements on a page, as well as modify the text of individual elements. In essence, scripts can construct and alter the content of a document at run time. For example, a script can scan the elements of a page and, using

dynamic content, insert a table of contents at the beginning of the page. Furthermore, the table of contents can be made live, using links to bookmarks.

Unlike other approaches, which restrict content changes to load time only (using syntax such as document write), these changes can be made at any time, even after the entire document has been loaded.

Filter, Transition, and Animation Controls

Dynamic HTML includes animation and multimedia controls that can be used to apply visual effects to elements on a page or to the entire page. These controls support filters, animation and transitions. Transitions can be used for elements on a page, or for transitions between pages.

Furthermore, these controls can take advantage of new multimedia and animation services delivered with Microsoft Internet Explorer 4.0. ActiveX Control and Java developers can take advantage of these services to implement additional multimedia or animation effects.

Data Awareness

Many HTML pages are based on data, regardless of whether that data is stored in databases or files. Dynamic HTML incorporates several features to integrate data with native HTML elements. These features make HTML a better environment for displaying and collecting data. Microsoft Internet Explorer 4.0 includes the following data awareness features:

- **Automatic generation of table rows from data records.** By binding a table to a data source, Dynamic HTML can automatically create a table row for each record in the data source.
- **The table expansion is dynamic.** The user can view the page while the table is still being rendered. This contrasts with conventional server-generated tables that must be entirely generated before the page is sent to the client.
- **The table can be regenerated dynamically** (to support sorting and filtering) on the client without requiring the server to send additional data. This provides a new level of user interactivity without requiring round trips to the server.
- **Binding of HTML elements to a specific record.** By designating a record as the current record and binding fields from the record to elements on the page, data from the current record is displayed as part of the HTML in the document.
- **Data-bound form fields.** HTML forms can be created using intrinsic controls and compliant ActiveX Controls. These controls can be bound to fields in a record. Whatever the user enters in the control is then pushed into the record in the data source control. Upon user command (e.g., pushing a button), the data source control then uploads the data to the HTTP or database.

Dynamic HTML data awareness is implemented using an open architecture. Within each data-aware page is a data source control. A data source control is an invisible ActiveX Control that knows how to communicate with a data source (e.g., database). Dynamic HTML intrinsically understands how to bind HTML elements to fields in the data source control. Dynamic HTML will include three data source controls to access comma-delimited data in files, SQL data in Microsoft SQL Server™ and other open database connectivity (ODBC) sources, and Java database connectivity (JDBC) data sources. ActiveX Control and Java

Applet developers can implement additional data source controls to communicate with other data sources.

Dynamic HTML's data awareness functionality provides a rich set of options to allow the author to use native HTML to provide users with the ability to manipulate and input data efficiently, with minimal load on the server. The result is faster, richer and more interactive pages.

↑ Top of page

Printer-Friendly Version Send This Page Add to Favorites Comments

Manage Your Profile | Contact Us | Newsletter

What is Dynamic HTML?

by Nadav Savio *2 Oct 1997*

Nadav Savio runs Giant Ant Design, a human-centered interactive design shop. When he's not cursing non-standard browser implementations, you can find him drawing futuristic utopias or updating his interface-design weblog, antenna.

Page 1

Q: Dear Webmonkeys,
I keep hearing about "dynamic HTML," but I don't understand what it is. I really think you should write about it.
- Fern

A: Dear Fern,

First of all, **dynamic HTML** as a self-contained thing-unto-itself is really just an idea. It is not any one specific technology (such as JavaScript or ActiveX). Nor is it a tag, a plug-in, or a browser.

Dynamic HTML (aka dHTML or DHTML) is one of the most exciting and useful things to happen to the Web in recent memory. It's a concept that has been enabled (to different extents in different browsers, of course) by a number of technologies, including JavaScript, VBScript, the Document Object Model (DOM), layers, and Cascading Style Sheets (CSS).

So, what IS it? Here's the short answer: Dynamic HTML is simply HTML that can change even after a page has been loaded into a browser. A paragraph could turn blue when the

mouse moves over it, or a header could slide across the screen. Anything that can be done in HTML can be redone after the page loads.

So, how can HTML be changed after it's been downloaded? There needs to be some way to tell the browser to change it, which brings us to the technologies that make up DHTML:

- **Dynamic HTML is client-side scripting**
  People have been using client-side scripting languages (JavaScript and VBScript in particular) to change HTML for a long time. If an image changes when you roll your mouse over it, you're looking at an example of dynamic HTML. The 4.0 browsers from both Microsoft and Netscape allow more of a page's HTML elements to be accessible from within scripting languages. The mechanism whereby page elements (or document objects) are opened to scripting languages is called the Document Object Model.

- **Dynamic HTML is the DOM**
  In a sense, the Document Object Model is the real core of dynamic HTML. It makes HTML *changeable*. The DOM is the hierarchy of elements that are present in the browser at any given time. This includes environmental information such as the current date and time, browser properties such as the browser's version number, window properties such as `window.location` (the page's URL), and HTML elements such as `<p>` tags, `divs`, or tables. By exposing the DOM to scripting languages, browsers enable you to access these elements. While some elements such as the time of day can't be changed themselves, they can be used by scripts to modify other elements.

  As Taylor has previously described, there was a Document Object Model before Internet Explorer 4.0, but earlier implementations pale in comparison. With IE 4, it is possible to access any part of the HTML on your page using any scripting language that runs in the browser.

  (Although Netscape's DOM is much more limited than IE's, you can put the part of your page you want to change in a `layer` container and then change the layer. And Netscape says it will fully support the W3C's DOM specification in future versions of the browser.)

  The part of the DOM that specifies which elements can trigger changes is the event model. Events are things like moving the mouse over an element (`onmouseover`), loading a page (`onload`), submitting a form (`onsubmit`), clicking on a form input field (`onfocus`), and so on.

- **Dynamic HTML is CSS**
  Because they are part of the DOM, CSS properties are accessible to scripting languages, and it is therefore possible to change almost anything about the way a page looks. By changing the CSS properties of a page element (such as its color, position, or size), you can do almost anything bandwidth and processor speed permit.

To sum all this up: CSS (and plain old HTML) is *what you change,* the DOM is *what makes it changeable,* and client-side scripting is *what actually changes it.* And *that's* dynamic HTML.

To learn more, take Taylor's dHTML Crash Course.

Explore Creativity and Control With Dynamic HTML

Adapted from article by William R. Stanek in PC Magazine, January 20, 1998

Dynamic HTML lets Web authors do more--and do it more precisely--than ever before.

To remain static on the Web is to court obsolescence. Not only do we expect sites to change daily, but we also expect them to provide ever-increasing levels of interactivity, along with special effects that rival the most sophisticated PowerPoint presentations. And we expect a constant flow of new and better technologies to supply the means. Dynamic HTML is the latest of these technologies and promises to enhance the design and capabilities of our Web pages. As has become familiar in the Internet arena, however, the two major players, Microsoft Corp. and Netscape Communications Corp., have developed separate and entirely different versions of the technology. As a result, your wonderful dynamic Web site may work with one browser but not the other.

So as a Web publisher, what do you do? Do you choose one version of dynamic HTML over the other? Or is there a way to create dynamic pages that will work with Microsoft Internet Explorer *and* Netscape Navigator? To help you decide, we will look at both companies' versions of dynamic HTML with an eye to creating dynamic pages that will work with both browsers.

With dynamic HTML, you can create Web pages with eye-popping special effects, animation, and much more without relying on server-side scripts, database engines, or hundreds of lines of complicated markup code. One of the key design goals in creating dynamic HTML was easing the complexities involved in interactive multimedia presentations on the Web. An important part of that goal was building the necessary support framework into the browser, which is exactly what Microsoft and Netscape did with Version 4.0 of their respective browsers. The result is that you don't have to rely on ActiveX controls, Netscape plug-ins, or other helper applications to achieve special effects, animation, or anything else that dynamic HTML enables.

Dynamic content is not entirely new with the new browser versions. Client and server applications have supported dynamic content for some time now. If you use any of Microsoft's Active servers, such as the Internet Information Server, you can use active server pages to tailor your content to the needs of individual users. Similarly, Netscape's Enterprise or FastTrack servers let you use server-side scripting to individualize content. Both companies' technologies also allow you to use client-side scripts to create highly interactive dynamic pages. The scripts could be written in JavaScript, JScript, or VBScript, depending on the client browser you are developing for.

What these and other solutions for creating dynamic content have in common is complexity. To create dazzling Web pages full of bells and whistles, you need to perform some pretty virtuosic programming feats. Thanks to dynamic HTML, you can create the same Web pages with minimal coding. A page that uses dynamic HTML to make live updates doesn't look much different than its static counterpart, but achieving the same effect with server-side scripting would be a noteworthy feat.

The key to dynamic HTML in both Internet Explorer and Navigator is a live update mechanism that allows a browser to modify sections of a Web page in the background. Once the page has been modified, the browser reformats it as necessary and displays the changes. Anyone viewing the page sees the updates instantly and doesn't have to wait for the browser to reload the page or access another page. The browser makes the changes without ever having to go back to the Web server for additional content.

Live updates can be triggered by user- and browser-driven events as well as by timers. Whenever a user moves the mouse over an element in a Web page or clicks on a button, an event is triggered that can be

handled by dynamic HTML. Browser-driven events are triggered by internal actions, such as when the browser finishes downloading the Web page. With timers, you use a script or a live media feed to update content on the page.

Of course, events and timers aren't new with dynamic HTML. What's new is how they are handled. Instead of relying on Java applets, ActiveX controls, or Netscape plug-ins, the browser can handle the live updates directly. That said, both Microsoft and Netscape are quick to point out that dynamic HTML can be used with embedded objects, which ideally you would use to manipulate dynamic HTML in ways that can't be handled directly by the browser, or to perform other complex tasks.

Two Approaches: Microsoft and Netscape

Unlike tables, frames, and other recent Web page innovations, Microsoft's dynamic HTML doesn't focus on new tags and complex markup. One of the overriding design considerations for Microsoft was to make dynamic HTML a natural extension of existing technologies, allowing Web publishers to do more and go further with the tools they already have. Because of this, Microsoft dynamic HTML is easy to learn and use.

Microsoft's dynamic HTML is implemented as an extension of the Internet Explorer object model and is available only in Version 4.0 or later. The Internet Explorer object model provides the core functionality of the browser and is separate from the object models for scripting, ActiveX, and Java, which allows Microsoft to add functionality to the browser yet easily maintain compatibility between existing frameworks. That approach is the reason you can readily manipulate the features of dynamic HTML using applets, controls, and scripts, for which you'd likely use JScript. In a way, Microsoft's dynamic HTML is a scriptable combination of HTML and Cascading Style Sheets (CSS). For example, you can use JScript to change dynamically the contents of any HTML element in a Web page. You can also use JScript to change dynamically the style associated with any element in a Web page.

In short, Microsoft's implementation has focused on adding functionality to the browser through HTML and CSS extensions. But that's not all there is to it. In a head-to-head comparison, you will find quite a few innovations that are quite remarkable.

Netscape's dynamic HTML is also powerful, but it is implemented in a very different way involving a combination of new HTML tags, scripting extensions, and style-sheet extras. The focus is on new ways of doing things rather than extending existing technologies, so there are usually several different ways to implement the same functionality. Beginners may find that confusing, but more experienced Web publishers will likely welcome the opportunity to choose their own path.

As you set out to work with Netscape's dynamic HTML, keep in mind the enhancements are available only in Navigator 4.0 or later. Because dynamic HTML extends the browser object model, it is possible to use it with Java applets and Netscape plug-ins, though your primary means of working with the technology is through JavaScript. New JavaScript features even allow you to define style sheets using a JavaScript syntax.

A Head-to-Head Comparison

While Internet Explorer and Navigator implement dynamic HTML in different ways, there are places where the technologies converge. As I dissected both companies' implementations for this article, I came up with a list of seven innovations that the combined technologies offer:

- dynamic content
- dynamic styles

- element positioning
- animation
- multimedia effects
- dynamic fonts
- data binding

As we discuss them, I'll present four examples that illustrate some of the points. Click on "live examples" on the left to see the examples in action, or click on "code examples" to see the HTML.

Dynamic content and style changes

Dynamic content and style extensions allow you to change the content and appearance of a Web page after it is loaded. Because all elements in the Web page are scriptable with user- and browser-driven events, you can make every element in the page interactive. Users could click on a table header cell, for example, to get more detailed information, or they could move the pointer over a quotation to highlight it.

Microsoft takes the concept of dynamic content and style changes a step further by letting you completely replace elements on a Web page using a JScript method called outerHTML, which tells JScript to replace the element and its contents entirely. There's also a related method, innerHTML, which lets you replace the contents of an element.

Example 1 shows how you can replace elements in Internet Explorer. When you move the pointer over the text "See my e-mail address," the text is replaced with an address element that contains an actual e-mail address. Note the use of outerHTML to replace the paragraph element with the address element.

Netscape takes a much different approach to style and content changes. You can create and manipulate styles using the syntax defined in the Cascading Style Sheets Level 1 specification and in Netscape's own JavaScript syntax. (The special JavaScript syntax is one of the reasons Netscape's dynamic HTML may be confusing for beginners.)

Navigator allows you to use a traditional style sheet, like this:

```
<STYLE TYPE="text/css">
H1 { font-style: italic;
font-size: 25 pt;
color: blue
}
</STYLE>
```

Navigator also allows you to use a JavaScript style sheet, like this:

```
<STYLE TYPE="text/javascript">
tags.H1.fontStyle =
"italic"
tags.H1.fontSize = "25 pt"
tags.H1.color = "blue"
</STYLE>
```

When you compare the code samples, you will note several important differences in the second sample. The TYPE attribute references the JavaScript syntax instead of the CSS syntax. Styles are referenced via the tags object and their unique designators. (To extract a designator for any HTML element, remove the <> symbols from the element's begin tag.)

Note also the way CSS attributes are converted to JavaScript properties. While one-word attributes generally have identical property names, multiple-word attributes are combined by removing the hyphens between words and capitalizing the first letters of the second and successive words. For example, the color CSS attribute has a related color property, but the font-size CSS attribute becomes the fontSize property (as shown in the previous code sample).

It's probably a good idea to forget about the JavaScript style-sheet syntax and focus instead on the facts that styles can be manipulated through JavaScript and that CSS attributes are available as JavaScript properties, which together allow you to script dynamic styles in much the same way as you would with Microsoft's dynamic HTML.

Example 2 shows how you can dynamically change the style of an element within a Web page. Initially, the division element has a red background. When you move the pointer over the division, the background turns yellow. When you move the pointer away, the background turns green. Using similar techniques, you can also update content within a page dynamically.

Element Positioning and Animation

The positioning extensions for CSS allow you to place an element precisely on the screen by assigning the element specific *x*- and *y*-coordinates. Because you can assign a coordinate on the *z*-plane, you can also stack elements on top of each other. For example, you can place an image behind any type of text element by assigning it the same screen position and giving it a lower index on the

z-plane.

While valid *z*-index ranges for Internet Explorer include positive and negative integers, Navigator *z*-indexes must be 0 or a positive integer. The *z*-index of normal page elements is 0. Give a table a *z*-index of 1 and it will seem to float above other elements on the page. By making the table react to events, such as a mouse click, you can provide a way for readers to fold the table away, revealing the elements behind it.

Similar techniques can be used to create an animation. When a user clicks on an element or moves the pointer over it, you can shift the element's position. Changing an element's position is as easy as changing its style or content. However, if you don't control the speed of the animation, your element may zip across the page at light speed. To slow things down, use a timer. Both Internet Explorer and Navigator support a timer method called setInterval, which has the following syntax:

```
setInterval("methodToCall", timeInterval)
```

where methodToCall is the actual method you want to call and timeInterval is the delay in milliseconds between method calls.

Figure 3 shows how you can use a timer in Internet Explorer. If you run the demo on the *PC Magazine* Web site, you'll see an image slide across the browser window. Because the image's *z*-index is set below other elements on the page, the image moves behind the header text. If you try to implement a similar animation in Navigator, you can't use a negative *z*-index, but there is a work-around--assign a higher *z*-index to the elements you want the image to float behind.

As you'll soon discover if you work with Microsoft's dynamic HTML, positioning and moving groups of elements is best handled by making the elements part of a page division or span, so that you can address the group of elements as a whole rather than having to deal with each individually.

Netscape, of course, handles this differently, introducing a new set of tags designed for manipulating blocks of content within your Web page. In Navigator, each group of elements you want to manipulate can be defined as a document layer. Layers can be positioned and sized just like any other element. By specifying different *z*-plane indexes, you can stack layers on top of each other and other elements. Layers can also be resized and moved using methods of the layer object.

Example 4 shows how you can create and animate a layer. Move the pointer over the layer and the layer shifts to the left. Take the pointer off the layer and the layer moves down. As you take a look at the code, note the use of the <LAYER> tag.

In many ways, Netscape layers may remind you of frames. That's because the two elements have a lot in common. Both layers and frames have a wide assortment of attributes and can be manipulated using properties and methods of their corresponding JavaScript objects. Just as there is a <NOFRAME> tag, there is also a <NOLAYER> tag, which can be used to define content for browsers that don't support layers. There's even an <ILAYER> tag used to define internal layers within a page. Despite the fact that this technology is complex and can be confusing, layers are one of the most compelling reasons to use Netscape's dynamic HTML.

Multimedia Effects

In the area of multimedia effects, Microsoft is the clear winner. Its dynamic HTML allows you to use special effects and transitions without ever having to write a script. Scriptless multimedia effects are possible because Internet Explorer supports filters that can be invoked as CSS properties. You can think of these filters as a set of preprogrammed behaviors you can apply to controllable elements in a Web page. Another name for a controllable element is an HTML control. With Internet Explorer 4.0, Microsoft has greatly expanded the available HTML controls, which now include images, marquees, input fields, buttons, text areas, tables, individual table elements, page divisions, and page spans.

If you have ever seen an animated GIF that faded a graphic image in and out and wished you could do that without the byte-hogging graphic, you will love the blend-transition filter. Using this filter, you can fade any controllable element. You can fade out an image simply by specifying the blend-transition filter in the image's style attribute. The following image will fade for 5 seconds:

```
<IMG ID="image1" SRC="sun.gif"
STYLE="filter: blendTrans (duration=5.0)">
```

Another cool filter is the reveal-transition filter. You can use this filter to create transitions both within pages and between pages. A transition within a page allows you to reveal or hide an image, table, or other controllable element. A transition between pages allows you to create a slide-show presentation as readers navigate through your Web site.

Other filters you'll want to use include a set of visual filters you can use to create a drop shadow for a button, flip an image, or make a table seem to glow. The table "Filters Available in Microsoft's Dynamic HTML" summarizes the available filters and their uses.

While the main reason to use filters is to reduce complexity and make pages load faster, you can easily combine multiple effects and transitions with scripts, and you can use filters with some ActiveX controls. The main thing to remember with filters is that they cannot be used with any elements or controls that run in their own window, such as an internal frame or a windowed control.

Unfortunately, there is no functional equivalent of Internet Explorer filters for Navigator. Although you

could argue that similar effects and transitions can be accomplished through scripts, plug-ins, and applets, the amount of time and effort necessary to mimic these multimedia effects make the task too difficult for the average Web publisher.

Dynamic Fonts

With dynamic fonts, the idea is that you don't have to worry about whether a reader has a font you're using in your Web page. All you need to do is publish the source for a font at your Web site and insert markup into your Web page that tells the browser to download the font. When the page is accessed, the reader's browser retrieves the font and displays the page exactly as it was meant to be seen. Although the font is not permanently installed in the end user's computer, it is temporarily available for the specified Web page.

As you'll quickly discover if you delve into dynamic fonts, theory and practice are two different things. Dynamic fonts are not easy for novices to implement. Before doing anything, you may well need to install an authoring tool for creating font definition files based on the fonts installed on your computer. Generally, you use the tool to create a font definition file for each font family that you want to use as a dynamic font. Both Internet Explorer and Navigator provide support for dynamic fonts.

Data Binding

If you work with database records, or if you need to extract information from document files, you will want to take a close look at *data binding,* which is supported by Internet Explorer but not by Navigator. With data binding you can tie elements in your document to a data source. When the page is loaded, the current information from the data source can be read and dynamically inserted.

One way to use data binding would be to perform routine database searches. If you tally invoices every day and create a report summary from the results, you could use data binding to extract records from your database and automatically create a table based on the records. Once the records are stored in the table, you can filter and sort the data without ever having to go back to the database or the Web server.

Summing Up

Our comparison of the Microsoft's and Netscape's implementations of dynamic HTML reveals many differences between these technologies. In some ways, Microsoft's is clearly superior, especially when it comes to multimedia effects and data binding. Still, the versatility of Netscape's implementation, in particular unique features such as layers and JavaScript structured style sheets, should not be overlooked.

Ultimately, the dynamic HTML version you choose will depend largely on where and how you plan to use dynamic Web pages. If you are publishing on a corporate intranet, you may well be able to design your pages based on a particular browser. Beyond the intranet environment, you have no control over which browser is used to view your Web site. Creating cross-browser dynamic HTML pages is a tall order, because pages created for one browser won't work with the other. For the moment, the best idea may be to create dynamic HTML pages that won't cause problems in noncompliant browsers. Here are some pointers that may help.

- Browsers ignore tags and attributes they don't understand, which means you can use dynamic HTML tags and attributes without worrying about what will happen if another browser encounters them. Thus, you can safely use the Netscape <LAYER> tag in a Web page that will be viewed with any other browser. Internet Explorer, previous versions of Navigator, and other browsers ignore the Netscape layer and all the assignments within it.
- The navigator object provides the most efficient way to prevent noncompliant browsers from

executing scripts with dynamic HTML extensions. You can use properties and methods of this object to verify browser type and version. For example, to ensure visitors to your site are using Version 4.0 of Internet Explorer or Navigator, you can use the following control:

```
<SCRIPT LANGUAGE="JavaScript">
if (navigator.userAgent.indexOf ("Mozilla/4.0") != -1) {
//execute version 4.0 code or calls
} else {
//perform non-version 4.0 code or calls
}
</SCRIPT>
```

Because the userAgent property of both Internet Explorer 4.0 and Navigator 4.0 contains the text *Mozilla/4.0,* both browsers will execute code specified in the if statement. To get Internet Explorer and Navigator to execute separate statements, you need to use the appName and appVersion properties:

```
<SCRIPT LANGUAGE="JavaScript">

if (navigator.appVersion.charAt(0)
== "4") {

if (navigator.appName ==
"Netscape") {
//perform Navigator version 4.0 code or calls
} else {
//perform Internet Explorer version 4.0 code or calls
}
}
</SCRIPT>
```

Now that you've divided and conquered, you can create dynamic pages that should work in both Internet Explorer and Navigator. Unfortunately, there's not a lot of common ground to exploit between the companies' technologies, so at this point you may well be coding everything twice. Still, dynamic HTML has a lot to offer, and the results will justify your efforts. If you are more patient, then HTML 5 may solve most of this dilemma.

JavaScript™ FOR DUMMIES®

3RD EDITION

by Emily A. Vander Veer

IDG Books Worldwide, Inc.
An International Data Group Company

Foster City, CA ◆ Chicago, IL ◆ Indianapolis, IN ◆ New York, NY

JavaScript™ For Dummies®, 3rd Edition

Published by
IDG Books Worldwide, Inc.
An International Data Group Company
919 E. Hillsdale Blvd.
Suite 400
Foster City, CA 94404
www.idgbooks.com (IDG Books Worldwide Web Site)
www.dummies.com (Dummies Press Web Site)

Library of Congress Control Number: 00-106297

ISBN: 0-7645-0633-1

Printed in the United States of America

10 9 8 7 6 5 4 3

3B/SY/QZ/QQ/IN

Distributed in the United States by IDG Books Worldwide, Inc.

Distributed by CDG Books Canada Inc. for Canada; by Transworld Publishers Limited in the United Kingdom; by IDG Norge Books for Norway; by IDG Sweden Books for Sweden; by IDG Books Australia Publishing Corporation Pty. Ltd. for Australia and New Zealand; by TransQuest Publishers Pte Ltd. for Singapore, Malaysia, Thailand, Indonesia, and Hong Kong; by Gotop Information Inc. for Taiwan; by ICG Muse, Inc. for Japan; by Intersoft for South Africa; by Eyrolles for France; by International Thomson Publishing for Germany, Austria and Switzerland; by Distribuidora Cuspide for Argentina; by LR International for Brazil; by Galileo Libros for Chile; by Ediciones ZETA S.C.R. Ltda. for Peru; by WS Computer Publishing Corporation, Inc., for the Philippines; by Contemporanea de Ediciones for Venezuela; by Express Computer Distributors for the Caribbean and West Indies; by Micronesia Media Distributor, Inc. for Micronesia; by Chips Computadoras S.A. de C.V. for Mexico; by Editorial Norma de Panama S.A. for Panama; by American Bookshops for Finland.

For general information on IDG Books Worldwide's books in the U.S., please call our Consumer Customer Service department at 800-762-2974. For reseller information, including discounts and premium sales, please call our Reseller Customer Service department at 800-434-3422.

For information on where to purchase IDG Books Worldwide's books outside the U.S., please contact our International Sales department at 317-572-3993 or fax 317-572-4002.

For consumer information on foreign language translations, please contact our Customer Service department at 800-434-3422, fax 317-572-4002, or e-mail rights@idgbooks.com.

For information on licensing foreign or domestic rights, please phone +1-650-653-7098.

For sales inquiries and special prices for bulk quantities, please contact our Order Services department at 800-434-4322 or write to the address above.

For information on using IDG Books Worldwide's books in the classroom or for ordering examination copies, please contact our Educational Sales department at 800-434-2086 or fax 317-572-4005.

For press review copies, author interviews, or other publicity information, please contact our Public Relations department at 650-653-7000 or fax 650-653-7500.

For authorization to photocopy items for corporate, personal, or educational use, please contact Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, or fax 978-750-4470.

Chapter 11

Framed Again!

In This Chapter

- Manipulating frames with JavaScript
- Creating a simple JavaScript page map
- Creating a collapsible JavaScript site map

Frames let you display multiple HTML documents inside a single browser window. Using JavaScript, you can create all kinds of sophisticated frame effects! Two of the most useful (and most common) frame effects, both of which you see in this chapter's examples, are

- Using frames to create a dynamic site map
- Using frames to display hyperlinked documents

By the time you finish this chapter, you know enough about how JavaScript interacts with frames to create your own cool frame effects.

Whether or not to include HTML frames in your Web site is a personal design decision. Some folks love frames, because not only do they allow you to create effective navigation structures (like the site map you see in this chapter), they also allow you to provide hyperlinks while discouraging users from surfing off to those hyperlinked sites. The downside? Frames can be complicated to implement, and some people dislike the fact that they "hide" URL information. To see the URL for a link opened in a frame, for example, you can't just click on the link; you must right-click and select Open in New Window (Internet Explorer) or Open Link in New Window (Navigator). If you *do* decide to implement frames, however, JavaScript can help you make the most effective use of them.

One Page at a Time

Scripted frames are a valuable addition to any Web developer's tool belt. Using a combination of HTML frames and JavaScript, you can present a static, clickable table of contents on one side of a page; then, on the other side of the page, you can present the text that corresponds with each table of contents entry. Check out Figures 11-1 and 11-2 to see a table of contents. Figure 11-1 shows my site which has an example of a simple framed table of contents on the left side of the page and content on the right.

One of the benefits of frames is that they allow you to display different HTML files independently from one another. So, for example, as Figure 11-2 shows, the left frame stays visible — even if the user scrolls the right frame. Plus, clicking on a link in the frame on the left automatically loads the appropriate content in the frame you see on the right.

This approach, which I explain in the following sections, helps users navigate through the site quickly and is very useful for organizing small sites — or even larger sites that contain mostly text.

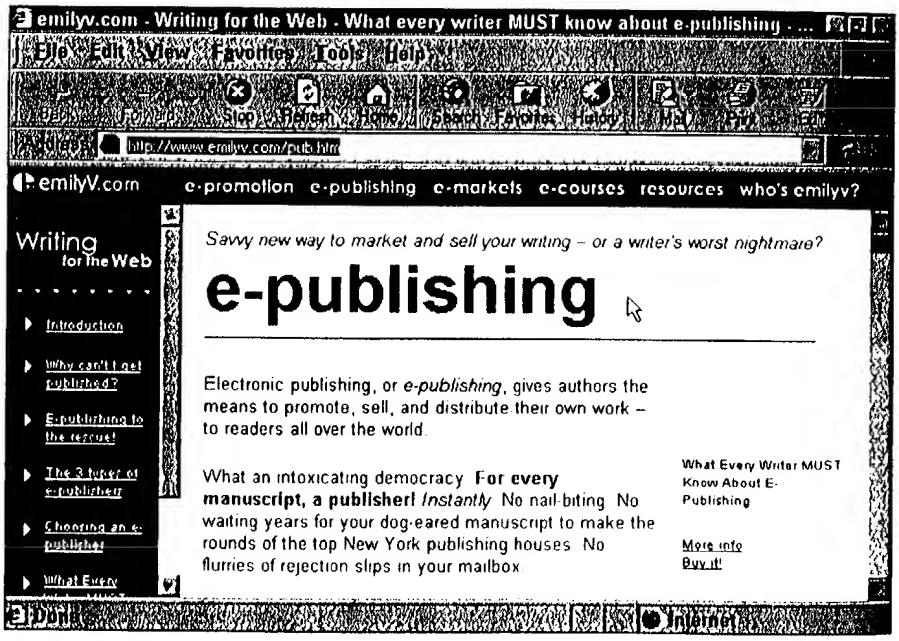


Figure 11-1: Using frames to split content text from a site index.

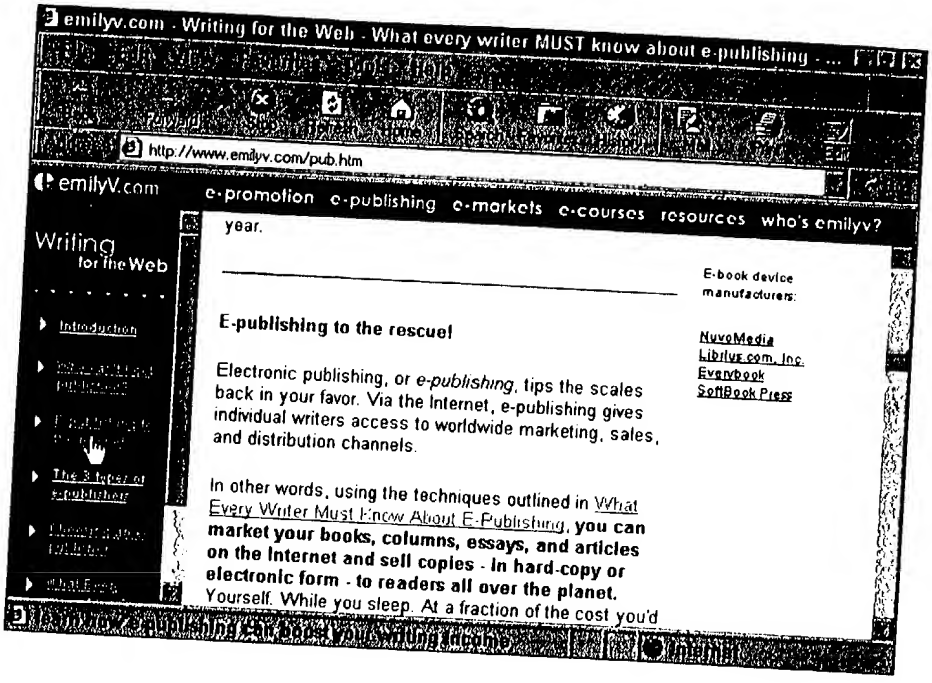


Figure 11-2: One frame jumps while the other stays put.

Constructing frames

A `frame` is a special type of `window` that you create using HTML tags and manipulate using JavaScript and the document object model. Since this book doesn't focus on HTML, I don't go into great detail on creating HTML frames; instead, I show you the basic syntax you need to know to understand how JavaScript and the document object model fit into the picture. (If you want to know more about creating HTML frames, you might want to pick up a copy of *HTML for Dummies*. I highly recommend it!) Listing 11-1 shows you an excerpt of the code you need to create frames, using the htm files to hold the two frames together. list1101.htm pulls together pub_l.htm (left frame's table of contents) and pub_c.htm (right frame's content).

You can view the complete working example of the code presented in this section from the companion CD: list1101.htm, pub_l.htm, and pub_c.htm.

Don't fence me in!

Just as you can display other folks' Web pages inside your frames, so those folks can display *your* Web pages inside *their* frames.

But in some cases, you may want to prevent your site from being framed. For example, say you spend weeks creating a beautiful, graphics-rich site optimized for a particular monitor size and screen resolution. Then, say I come along and add a link to your site from mine — but I choose to display your fabulous, pixel-perfect site by squeezing it into a tiny 2" x 2" frame! (Worse yet, I'm a cat lover, so I surround the 2" x 2" frame with an image of my beloved Fifi — so your site appears to be peeking out of my cat's mouth!)

To prevent other sites from displaying your document in a frame, add the short script you see below to your document's head.

```
<HEAD>
  . . .
<SCRIPT LANGUAGE="JavaScript">
<!-- Start hiding from non-
   JavaScript-support browsers
// The following JavaScript
   code reloads a page so
// that it "breaks out" of its
   container frame, if one
   exists.

// If this page has been loaded
   into a frame...
if (top != self) {
   // Replace the original
   "framing" page with the
   framed page
   top.location.href = loca-
   tion.href;
}
// Stop hiding -->
</SCRIPT>
  . . .
</HEAD>
```

Listing 11-1: HTML Syntax for Creating Index and Content Frames

```
  . . .
<FRAMESET COLS="125,*"
      BORDER="0"
      FRAMESPACING="0"
      FRAMEBORDER="NO">
// Defining the source file, name, and display details
      for the
// left frame
 <FRAME SRC="pub_1.htm"
NAME="leftnav"
SCROLLING="AUTO"
NORESIZE MARGINHEIGHT="0"
MARGINWIDTH="0"
LEFTMARGIN="0"
TOPMARGIN="0"
```

```
    TARGET="body">
      // Defining the source file, name, and display details
      // for the right frame
      <FRAME SRC="pub_c.htm"
             NAME="content"
             SCROLLING="AUTO"
             NORESIZE
             MARGINHEIGHT="0"
             MARGINWIDTH="0"
             LEFTMARGIN="0"
             TOPMARGIN="0"
TARGET="body">
      </FRAMESET>
```

Take a good look at the HTML code in Listing 11-1 to find the two frame definitions:

- `leftnav` (which corresponds to the HTML file pub_l.htm)
- `content` (which corresponds to the HTML file pub_c.htm)

The file pub_l.htm contains a list of content links (in other words, a table of contents); the file pub_c.htm contains corresponding text. Figures 11-3 and 11-4 show you what these two files look like when loaded separately into Internet Explorer.

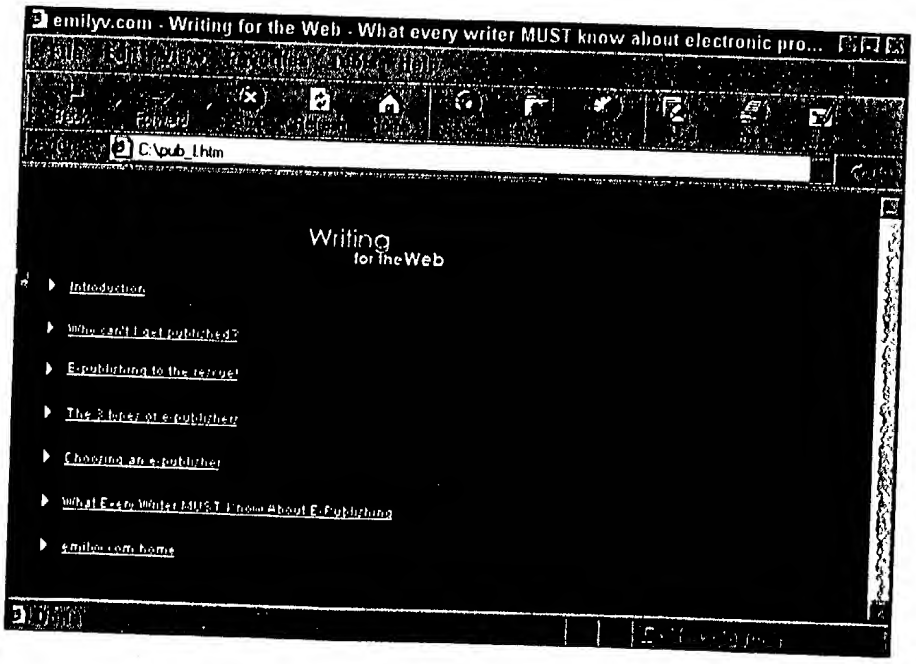


Figure 11-3: The table of contents as it appears by itself.

Looking at pages separately, before you put them into frames, helps you understand how to combine them for the best effect.

In the next section, you see how to connect the two files shown in Figures 11-3 and 11-4.

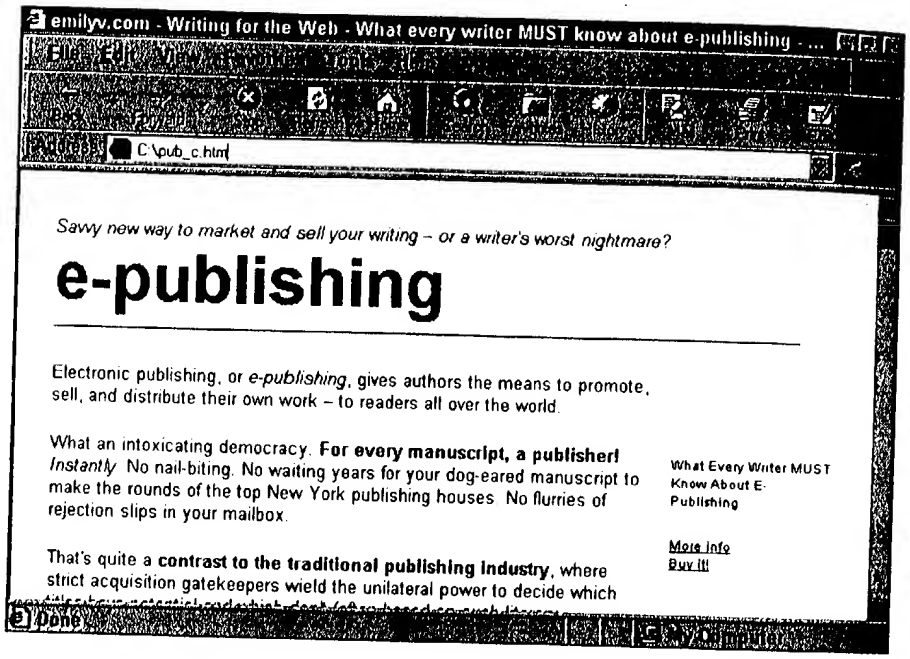


Figure 11-4: The text that corresponds to the table of contents shown in Figure 11-3.

Adding the wiring

In the example in this section, the content in the right frame reloads based on what a user clicks on in the left frame. So, naturally, the code responsible for the text reload can be found in the source code for the left frame, `pub_l.htm`. Take a look at the pertinent syntax as shown in Listing 11-2. This code snippet, from pub_l.htm, connects the table of contents links to the appropriate content.

Listing 11-2: Connecting the Index Links to the Content Headings

```
// When a user clicks the "Introduction" link,
// the anchor located at pro_c.htm/top loads into the
// frame named content.

<A HREF="pro_c.htm/top" TARGET="content">
```

Right on target

When you create a link (or an anchor, area, base, or form) in HTML, you have the option of specifying a value for the TARGET attribute associated with these HTML elements. Valid values for the TARGET attribute include any previously named frame or window, or one of the following built-in values. (Listing 11-2 demonstrates an example of specifying the _top value for the TARGET attribute associated with a link.)

Value	What does it mean?
_blank	Open the link in a brand-new window
_parent	Open the link in this window or frame's parent window/frame
_self	Open the link in this window or frame
_top	Open the link in the root window or frame

```
>Introduction</A>

. . .

// When a user clicks the "Why can't I get published?" link,
// the anchor located at pro_c.htm#can'tget loads into the
                frame
// named content

<A HREF="pub_c.htm#cantget" TARGET="content"
>Why can't I get published?</A>

<A HREF="pub_c.htm#rescue" TARGET="content"
>E-publishing to the rescue!</A>

<A HREF="pub_c.htm#types" TARGET="content"
>The 3 types of e-publishers</A>

<A HREF="pub_c.htm#choose" TARGET="content"
>Choosing an e-publisher</A>

<A HREF="pub_c.htm#epubGuide" TARGET="content"
>What Every Writer MUST Know About E-Publishing</A>

. . .

// When a user clicks the "emilyy.com home" link, a
// new page (home.htm) replaces the current page

<A HREF="home.htm" TARGET="_top"
>emilyy.com home</A>
```

Each of the links I define in Listing 11-2 contains a value for the TARGET attribute. With one exception, TARGET is set to content — the name of the right frame defined in Listing 11-1. Assigning the name of a frame to the TARGET attribute of a link causes that link to load in the named frame, just as you see in Figure 11-2.

You do need a variation for the exception: At the bottom of Listing 11-2, you see that the last link defined assigns a value of _top to the TARGET attribute. When a user clicks on the link marked emilyv.com home, the page changes to the contents of home.htm.

_top is a built-in value that translates to "whatever the top-level window in this window/frame hierarchy happens to be." (The sidebar "Right on target" in this chapter describes all the built-in values you can specify for the TARGET attribute.)

Specifying a value for TARGET that doesn't match either a) a previously defined frame name, or b) one of the built-in values you see in the sidebar "Right on target" causes the associated link to load into a brand-new window. So if you expect a link to open in a frame and it pops up in a new window instead, check your source code; odds are you've made a typo!

The example in this section shows you how to load the contents of one frame based on a user clicking a link in another. To load two frames based on a user clicking a link, you can create a JavaScript function similar to the following:

```
function loadTwoFrames(leftURL, contentURL) {

   // Loads the first URL passed in
   // into the container frame previously defined
   // as "leftNav" in an HTML file such as the one
   // you see in Listing 11-1

   parent.leftNav.location.href=leftURL

   // Loads the second URL passed in
   // into the container frame previously defined
   // as "content"

   parent.content.location.href=contentURL
}
```

Then pass the loadTwoFrames() function two URL strings; for example:

```
<A HREF="Javascript:loadTwoFrames('some.htm',
        'another.htm')">
```

or

```
<INPUT TYPE="button" VALUE="Load Two Frames"
        onClick="loadTwoFrames('some.htm',
            'another.htm')">
```

Creating Collapsible Indexes

Anyone who's used the Windows Explorer utility (see Figure 11-5) knows how useful a collapsible table of contents can be.

Clicking on any of the plus-sign icons shown in Figure 11-5 expands the associated directory; clicking on a minus-sing icon collapses the associated directory. The collapsible list shown on the left side of Figure 11-5 is a nifty, useful way to organize and present a lot of information in a logical, hierarchical manner.

Using a combination of free, pre-built JavaScript *libraries* and HTML frames, you can create similar collapsible lists for your Web site.

Checkin' out the library

When computer programmers talk about libraries, they don't mean quiet buildings where you can go to check out books! In programming parlance, a *library* is a bunch of functions — sometimes free, sometimes not — that you can reference and call from your own code.

To implement JavaScript libraries, you use them as `.js` files that you include in your own files with the help of the `SOURCE` attribute of the `<SCRIPT>` tag, as shown below (a complete working example can be seen in Listing 11-3):

```
//Including a JavaScript library with the SRC attribute
<SCRIPT LANGUAGE="JavaScript"
        SRC="someFile.js">
</SCRIPT>
```

```
<SCRIPT LANGUAGE="JavaScript">
...
// someFunction is defined in someFile.js
// so it is available for use here
// Calling a function defined by the someFile.js library
someFunction()
...
```

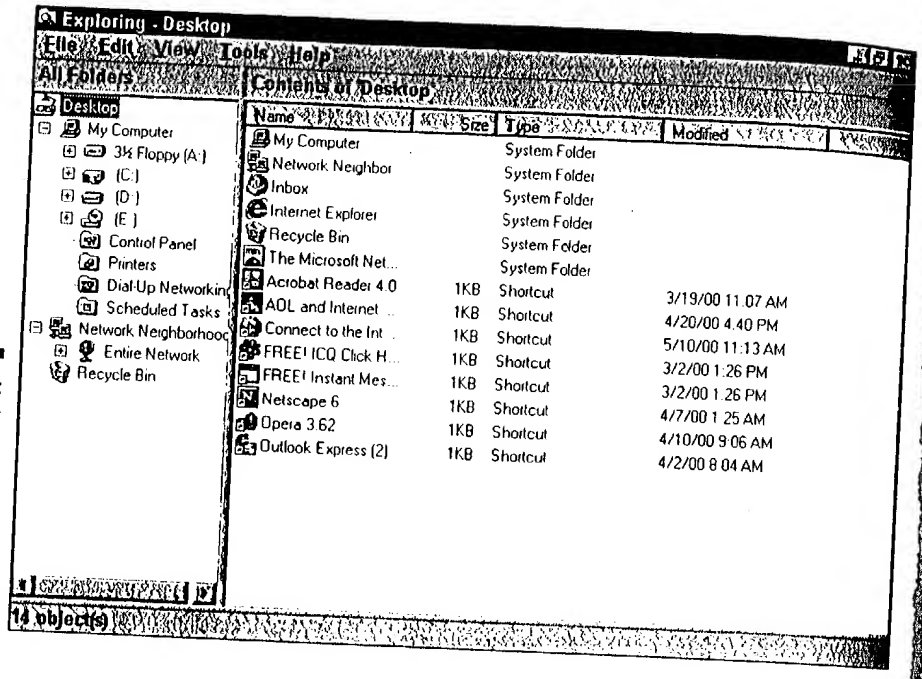


Figure 11-5: Explorer presents the contents of directories in neat collapsible lists.

The first script declaration in the preceding code, incorporates the file `someFile.js` into the HTML document. The second script declaration shows how to make a call to a function called `someFunction()`— a function defined by the `someFile.js` library. Organizing generic JavaScript code into reusable libraries offers these benefits:

- **Cuts down on errors.** After a JavaScript library is fully tested, you can reuse it with confidence. (Compare *that* to cutting and pasting JavaScript functions into each HTML file where you want to call those functions!)
- **Improves performance time.** A single copy of a JavaScript library downloads into memory where the JavaScript interpreter can access it, no matter how many pages (one document or a dozen) reference that library.

The good folks at Netscape have a library, list.js, currently available that offers collapsible list creation functions. (You see how to create a full-scale collapsible table of contents later in this chapter, in the "Outlining your site contents" section.) You can download a copy of list.js for free from the following site:

```
http://developer.netscape.com/docs/technote/dynhtml/collapse/
        index.html
```

Calling all code

Using a library can be simple or difficult, depending on how well the library developer documents the library!

Netscape documents fairly well the library example I demonstrate in this section, list.js. Listing 11-3 shows you an example of how you incorporate list.js and use the JavaScript functions it contains to create a collapsible list.

Listing 11-3: Using Netscape's list.js Library to Create a Collapsible List

```
<HTML>
<HEAD><TITLE>Creating collapsible lists with Netscape's
        list.js (from JavaScript For Dummies, 3rd
        edition)</TITLE>

// Including the list.js library.
<SCRIPT LANGUAGE="JavaScript1.2" SRC="list.js"></SCRIPT>

<SCRIPT LANGUAGE="JavaScript">
function init() {

  // Specify the list dimensions
  var width = 350, height = 22;

  // Create a new list.
  // List() function is defined and documented in list.js
  myList = new List(true, width, height, "#CCCCCC");

  // Add two items to the list
  myList.addItem("Computer");
  myList.addItem("Text editor");

  // Create a "sub" list
  subList = new List(false, width, height, "#CCCCCC");

  // Add three items to the new (sub) list
  subList.addItem("Netscape Navigator");
  subList.addItem("Internet Explorer");
  subList.addItem("America Online");

  // Attach the sub-list to the original list and
  // give it an entry.
  myList.addList(subList, "Browsers");
```

(continued)

Listing 11-3 *(continued)*

```
    // The list (complete with a sub-list) now exists
    // in memory, so display it on the screen
    myList.build(100,20);

}
</SCRIPT>
</HEAD>
<BODY ONLOAD="init();">
</BODY>
</HTML>
```

If you're new to working with code libraries, your first question is probably "How the heck do I know what the names of the functions are? Or what they do? Or what kind of values do I need to pass to them?"

The answer is you don't — unless the library developer tells you! Most code libraries have an accompanying text file that explains the functions that the library defines, how to call them, and a few working examples. In this case, Netscape kindly provides online examples and documentation (which you can find at the same site as the library itself):

```
http://developer.netscape.com/docs/technote/dynhtml/collapse/
        index.html).
```

Take a look at Figures 11-6 and 11-7 to see what the code in Listing 11-3 looks like in a back-level (4.0x) version of Netscape Navigator. When the user clicks on the right arrow next to `Browsers`, the code expands the list (in other words, displays the attached sub-list). The user can click the down arrow to collapse the list.

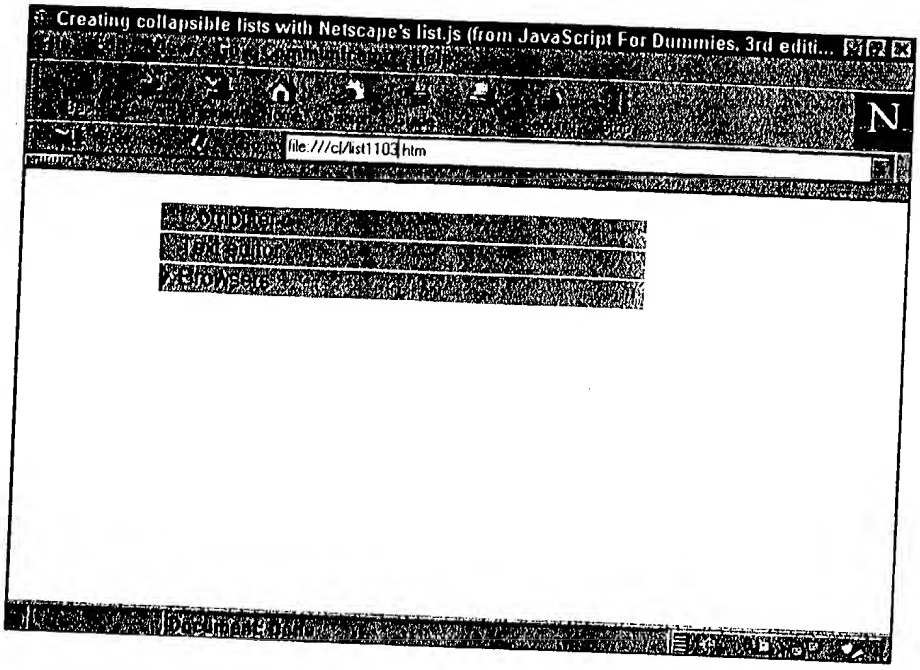


Figure 11-6: Using an arrow in the top-level list indicates more info's available.

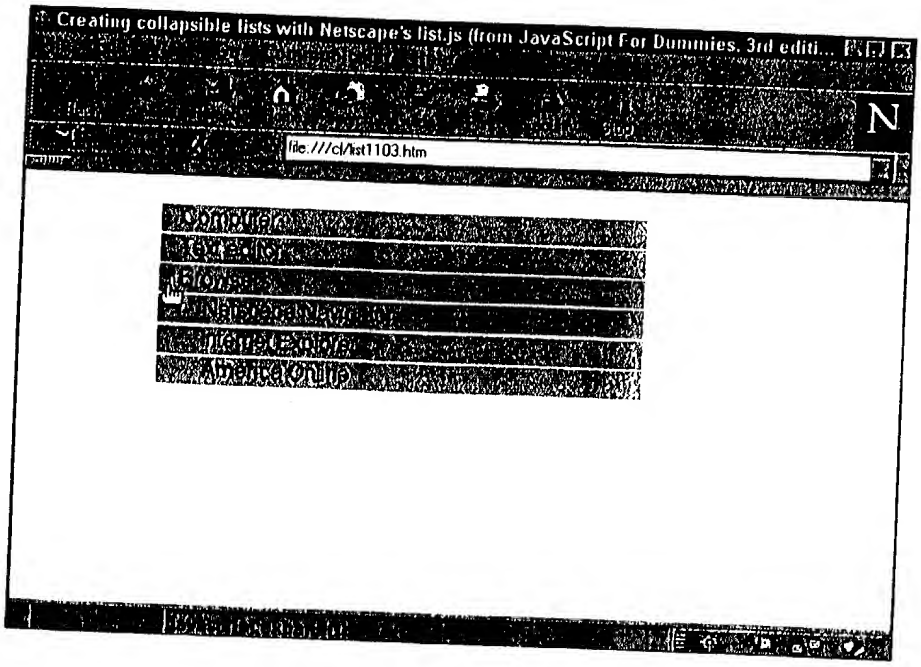


Figure 11-7: Expanding the list and toggles the up-down arrow.

Chapter 12

DHTML Dyn-o-mite!

In This Chapter

- Getting familiar with dynamic HTML (DHTML)
- Manipulating images and arrays
- Creating simple animations and slide shows

Dynamic HTML, or *DHTML*, refers to the collection of client-side languages and standards you use to create Web pages that change appearance dynamically, after they're loaded into a user's Web browser.

The languages and standards that contribute to DHTML include

- HTML (of course!)
- JavaScript
- Cascading style sheets
- The document object model (DOM)

Because this is a book about JavaScript, my focus in this chapter is on JavaScript and the document object model and how they combine to contribute to DHTML — in short, how you can use JavaScript to access and manipulate the DOM and create cool dynamic effects, including slide shows and simple animations.

While the examples in this chapter include HTML and cascading style sheet code, I don't spend a lot of time describing these two languages in-depth. If you're interested in finding out more about DHTML, including HTML and cascading style sheets, you might want to check out a good book devoted to these subjects. One worth checking out is "*Dynamic HTML For Dummies, 2nd Edition*," by Michael Hyman (IDG Books Worldwide, Inc.).

Chapter 3 describes the document object model and shows you how to access it; Appendix C presents both Internet Explorer's and Netscape Navigator's document object models.

Animation Sensation

Typically, when you see a cool animation on a Web page, you're looking at one of the following:

- **A Java applet:** Java applets are small software applications written in the Java programming language that your browser downloads from a Web server to your machine when you load a page.
- **A plug-in:** A plug-in is special software you can download that "plugs in" to your browser and allows an application to execute inside a Web page. Flash is one popular animation plug-in (from the good folks at Macromedia).
- **An animated GIF:** GIF stands for *graphics interchange format,* and it describes a special way of compressing image files. Regular GIF files are used to transfer images on the Web. Animated GIFs are a bunch of regular GIFs packaged together — much like those cartoon flip-books you may have had as a child, where each page contains a separate drawing. When you flip the flip-book pages (or load an animated GIF) those separate images flow from one to another to create an animated effect. Animated GIFs are a popular choice for Web-based animations because most browsers support them, no separate download is required (unlike plug-ins), and because they don't hog a lot of client resources (unlike some Java applets).

Having said all that, you can create simple animations with JavaScript, as well. You might want to do so for two very good reasons:

- Creating JavaScript animations saves your users time. (JavaScript animations don't require any downloads, either up-front or during animation execution, unlike plug-ins and applets, respectively).
- Creating JavaScript animations saves *you* the trouble of figuring out another programming language, such as Java, or learning how to use an animation construction tool, such as Macromedia's FireWorks.

The downside? Because JavaScript wasn't designed specifically to create animations, it isn't optimized for this purpose — meaning that specially-built functions and the compression techniques necessary for hard-core animation execution don't exist in JavaScript. In other words, JavaScript animations are best kept simple. Fortunately, many times, simple animations are all you need!

Another downside to using JavaScript to create animations lies in the incompatibilities between Netscape Navigator's and Internet Explorer's document object models — the very object models you must manipulate with JavaScript to create your animated effects. Netscape has announced changes in the way Navigator supports the animation-related portion of the DOM to coincide more closely with Internet Explorer's and the World Wide Web Consortium's standards, but to be sure your examples will work, you may want to test these in several versions of Navigator. The examples you see in this chapter work just fine in Internet Explorer. For more information on using the DOM and targeting dynamic HTML techniques for display in Navigator, visit DevEdge Online at this site

```
http://developer.netscape.com/docs/technote/
```

In this example, I show you how to create the simplest animation of all: an image that changes from its original view and then changes back again. Take a look at Figures 12-1 and 12-2 to see how my smiley changes, thanks to the recursive invocation of `setTimeout()`.

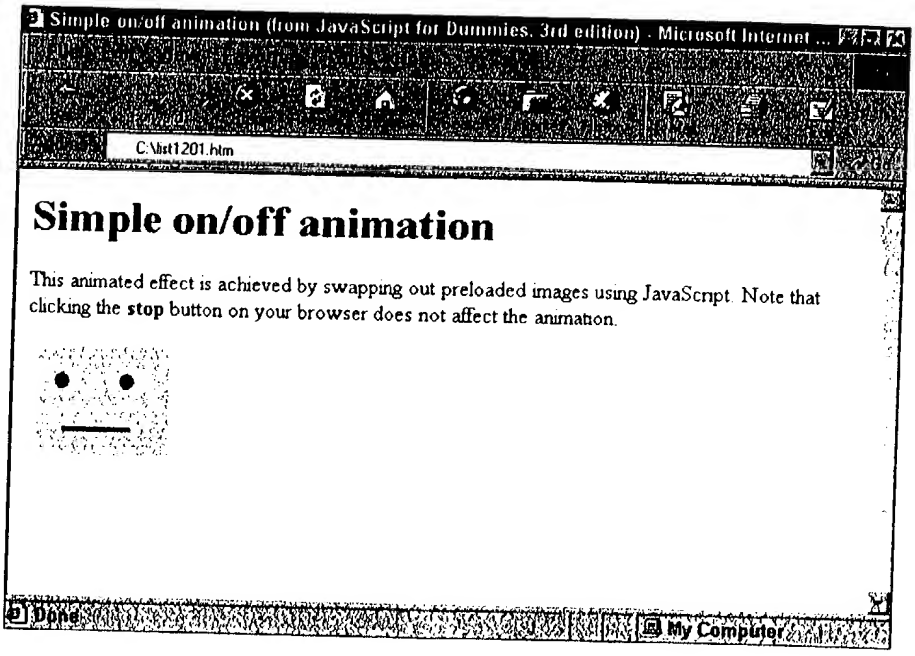


Figure 12-1: The neutral face . . .

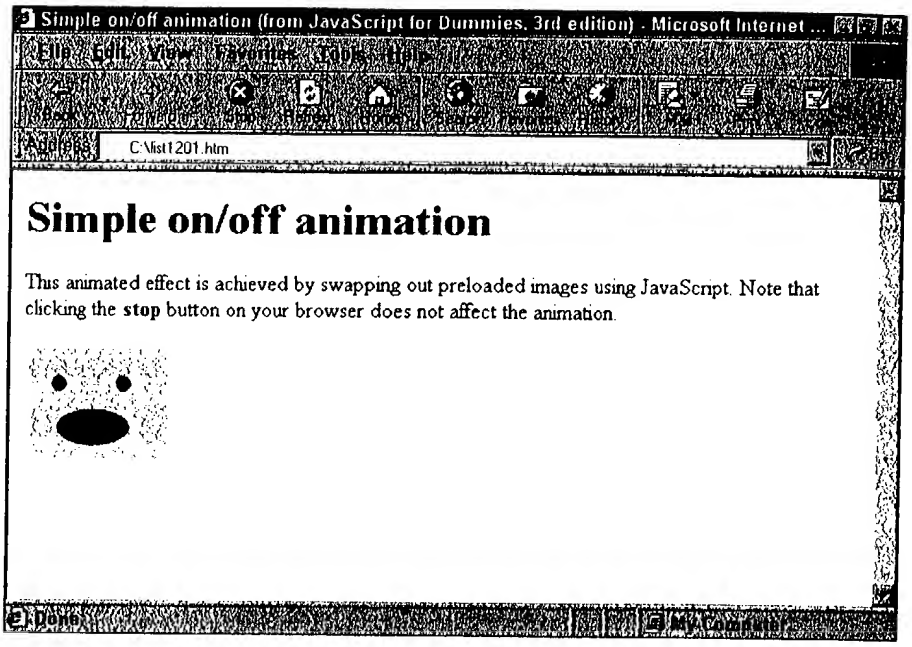


Figure 12-2: Changes into a surprised face every second.

The relevant code responsible for this simple animation is shown in Listing 12-1. Check out the simple on/off animation using image manipulation and the built-in JavaScript `setTimeout()` function. If you want to load and experiment with the animation example, load the file `list1201.htm` from the companion CD.

Listing 12-1: Creating a Simple Animation with JavaScript's `setTimeout()` Function

```
. . .
//Global variable declarations
var whichImg = 1
var nextImage

////////////////////////////////////////////////
// The swap() function replaces the image
// associated with the first input
// parameter (id) with the image specified
// for the second input parameter (newSrc)
////////////////////////////////////////////////

function swap(id, newSrc) {
    var theImage = findImage(document, id, 0);
    if (theImage) {
        theImage.src = newSrc;
```

```
    }
}

////////////////////////////////////////////
// This function exchanges the current image
// for the incoming parameter newImage;
// then it calls itself every second,
// passing itself a different newImage
// each time.  The result is a simple
// on/off animation.
////////////////////////////////////////////

function animate(newImage) {

        swap('animatedFace', newImage);

        if (whichImg == 1) {
            nextImage = "surprised.gif"
            whichImg = 0
        }
        else {
            nextImage = "neutral.gif"
            whichImg = 1
        }
//setTimeout() sets up the continuous swap that creates the
        animation
        setTimeout("animate(nextImage)", 1000)
}

. . .
// stop hiding -->
</SCRIPT>
</HEAD>

//The animate() function is called as soon as the page loads
<BODY onLoad="animate('surprised.gif')">
. . .
//The animation (image) dimensions are defined
<IMG NAME="animatedFace" SRC="neutral.gif" WIDTH="104"
        HEIGHT="80">
</BODY>
</HTML>
```

The JavaScript code in Listing 12-1 depends on two image files to create the animation:

- neutral.gif — This image of a yellow square contains two black "eyes" and a straight line for a mouth for the neutral look.
- surprised.gif — The image of the surprised "face." (Okay, okay, it's just a smiley face with a big circle for a mouth instead of a straight line. Artist's rendition! Artist's rendition!)

Here's the order in which JavaScript interpreter steps through the code in Listing 12-1 — a peek inside the interpreter's mind, as it were:

1. **The HTML `<IMG>` tag names and defines the animation placeholder frame (the spot on the page where the images appear alternately during the animation).**

   In the `<IMG>` tag, the name is `animatedFace`, and the dimensions are 80 x 104.

2. **As soon as the page loads, the `onLoad` event handler executes the `animate()` function and sends it the name of a source file (`surprised.gif`, to be exact).**

3. **The `animate()` function calls the swap() function to swap out the source file associated with the `animatedFace` placeholder frame.**

   Now, instead of the original `neutral.gif`, `animatedFace` holds `surprised.gif`.

4. **Using the globally defined variables `whichImg` and `nextImage`, the `animate()` function logs which image it just swapped out and queues up the next image by calling the `setTimeout()` function.**

   `setTimeout()` calls `animate()` every second, alternately passing `animate()` the `neutral.gif` and `surprised.gif` filenames.

Tickets to a Slide Show

Sometimes you want to set up a slideshow using JavaScript: a way for your users to click a button and see a different image, or "slide," without necessarily popping to another Web page.

Figures 12-3 through 12-5 show you the process of clicking a button to change the image from one view to the other.

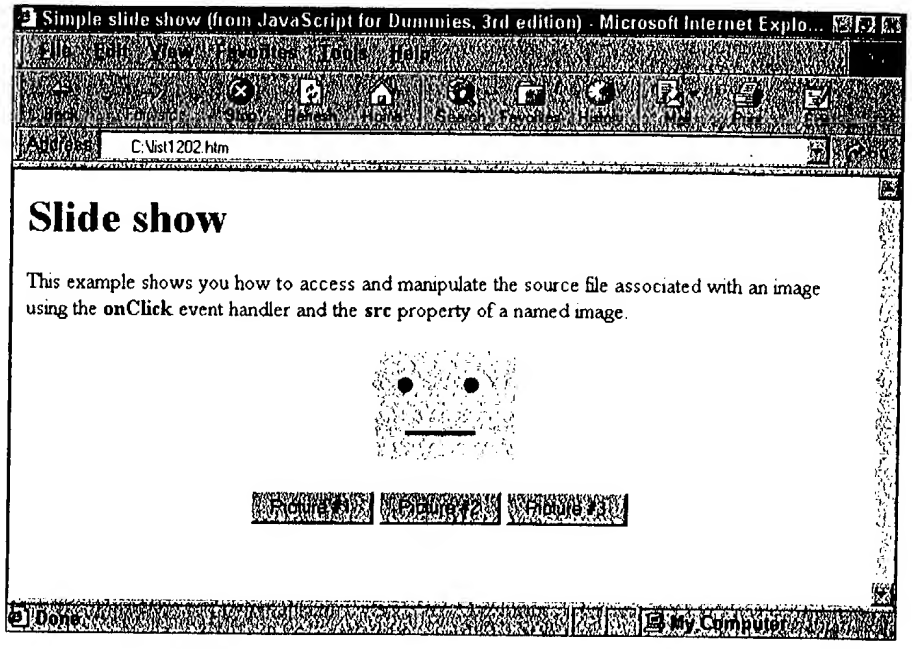


Figure 12-3: A neutral face appears by default, as soon as the page loads.

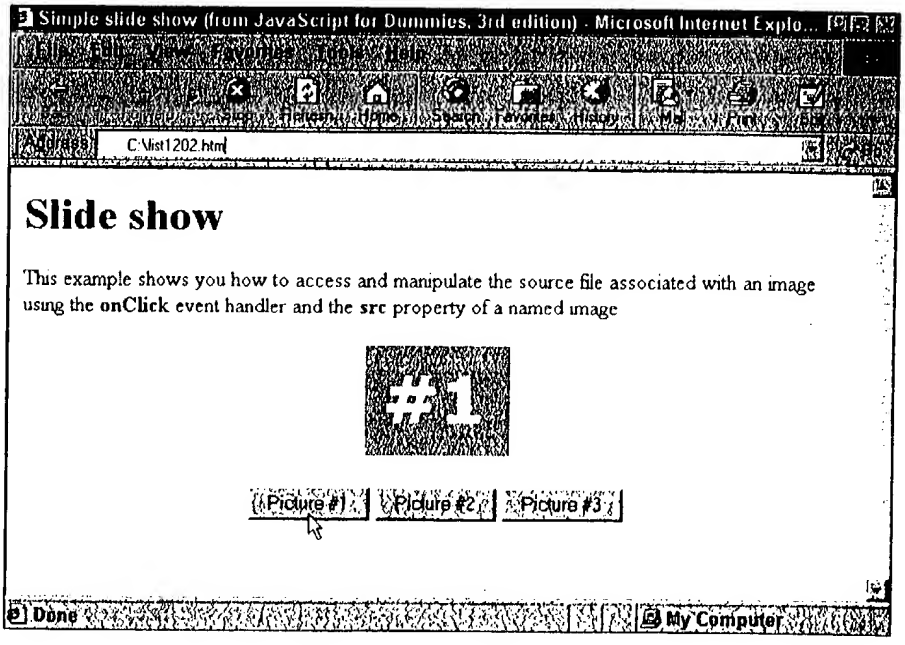


Figure 12-4: Clicking Picture #1 automatically displays the #1 image.

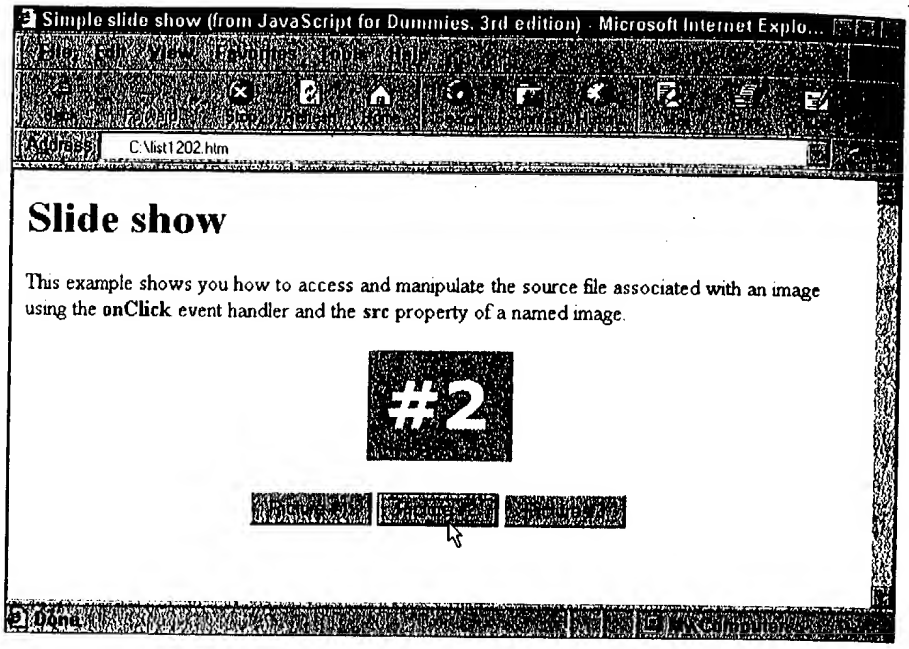


Figure 12-5: Keep clicking the buttons to cycle through the images.

The JavaScript code necessary to create the simple slide show example appears in Listing 12-2.

The code in Listing 12-2 is located on the companion CD in list1202.htm.

Listing 12-2: Creating a User-activated Slide Show

```
. . .

////////////////////////////////////////////////
// This swap() function constructs a file name
// based on the input parameter and then sets
// the slideshow image's source to that
// file name.
//
// Note: the initial image determines the size
// of the slideshow "frame".  Swapping to
// a larger image causes that larger image to
// be squeezed to fit the initial image "frame".
////////////////////////////////////////////////
function swap(newImage) {
```

```
    var fileName = newImage.toString() + ".gif"
    document.all.slideshow.src = fileName
}

// stop hiding -->
</SCRIPT>
</HEAD>

<BODY>
<H1>Slide show</H1>
This example shows you how to access and manipulate the
        source file associated with an image using the
        <B>onClick</B> event handler and the <B>src</B>
        property of a named image.

<!-- The initial image (a face) to display is specified
        here. -->

<IMG NAME="slideshow" SRC="neutral.gif" WIDTH="104"
        HEIGHT="80">

<P>

<!-- These three onClick event handlers call the swap()
        function to display the user-selected image -->

<INPUT TYPE="button" VALUE="Picture #1" onClick="swap('1')">
<INPUT TYPE="button" VALUE="Picture #2" onClick="swap('2')">
<INPUT TYPE="button" VALUE="Picture #3" onClick="swap('3')">
```

See the <IMG> tag near the bottom of the Listing 12-2 code listing? That <IMG> tag defines the initial image that displays when this page first appears, as shown in Figure 12-3, and names the placeholder for that initial image slideshow.

When a user clicks on any of the buttons — Picture #1, Picture #2, or Picture #3 — that button's onClick event handler springs into action and calls the swap() function, passing the swap() function the appropriate number: 1, 2, or 3. Inside the swap() function are just two lines of JavaScript code:

```
var fileName = newImage.toString() + ".gif"
document.all.slideshow.src = fileName
```

The first line creates a variable called fileName and then assigns to fileName a string based on the parameter sent to swap() from the onClick

event handler. (You must use the `toString()` method to convert the value of `newImage` to a string before you can tack on the ".gif.") After the JavaScript interpreter interprets this first line, `fileName` contains one of the following string values: `1.gif`, `2.gif`, or `3.gif`. (These file names correspond to actual GIF files located on the companion CD.) The second line of the `swap()` function assigns this new `fileName` to the built-in `src` property of the slideshow placeholder. (You specify a specific image placed in a document by navigating from the `document` object to the `all` object to the named `Image` object.)

Now You See It . . . Now You Don't!

A variation on the slide show example you see in the previous section is the ability to present an image to a user and then give that user the ability to hide the image. Hiding and showing images is easy using the built-in `visibility` property, as you see in Listing 12-3.

Listing 12-3: Giving Users the Option to Hide Images

```
var showing = true

//////////////////////////////////////////////////////////
// The toggle() function alternately hides and
// shows an element
//////////////////////////////////////////////////////////
function toggle() {
// If the image is visible, hide it
    if (showing == true) {
        document.all.toggledImg.style.visibility = "hidden"
        showing = false
    }
// Otherwise, the image is hidden, so show it
    else {
        document.all.toggledImg.style.visibility = "visible"
        showing = true
    }

    return true
}

// stop hiding -->
</SCRIPT>
```

```
</HEAD>

<BODY>
. . .
// You use the SPAN tag to create hide-able/show-able
            elements
<SPAN ID="toggledImg">
<IMG NAME="animatedFace" SRC="neutral.gif" WIDTH="104"
          HEIGHT="80">
</SPAN>
<P>
<INPUT TYPE="button" VALUE="toggle visible/invisible"
          onClick="toggle()">
</BODY>
</HTML>
```

To see what the code in Listing 12-3 looks like in action, see Figure 12-6, which shows the image using the `toggle()` function at `visibility = "visible"`.

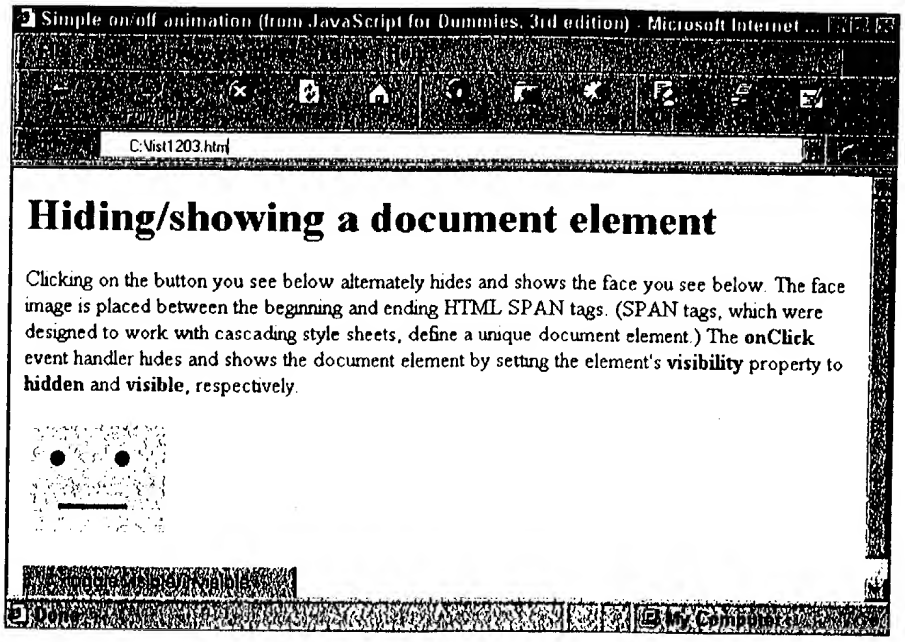


Figure 12-6: Now you see the image . . .

Figure 12-7 shows the opposite effect with `visibility = "hidden"`.

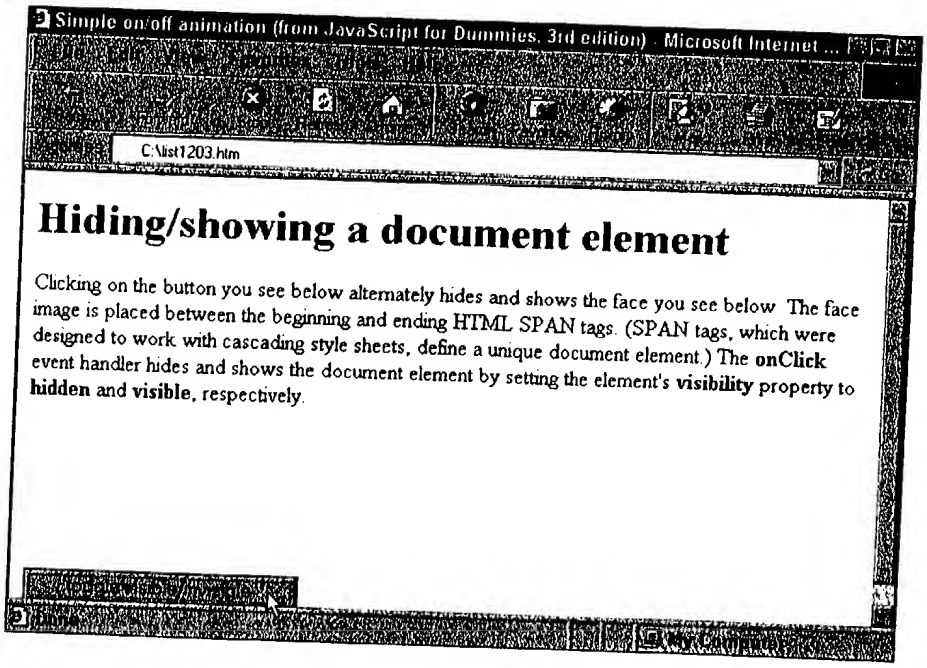


Figure 12-7: . . . Now you don't!

Three things to note about the code in Listing 12-3:

- The image is wrapped in a `span` element. (In other words, I place the the HTML `<IMG>` tag inside the beginning and ending HTML `<SPAN> . . . </SPAN>` tags.) The `span` element gives you the ability to show and hide the contents of a Web page — using the `span` element's `visibility` property.
- Changing an element's `visibility` property to `visible` immediately redisplays that element. Acceptable values for the visibility property are `visible`, `hidden`, and `inherit`. (Theoretically, a value of `inherit` tells the browser to use whatever visibility configuration is set for the element's containing element, but support for this property is not provided in the latest browsers.)
- Internet Explorer's object model implements the `visibility` property as part of the `style` object, which in turn is part of a `span` element (in this case, the `span` element named `toggledImg`), which in turn is part of the `all` object, which in turn is embedded in a `document`:

```
document.all.toggledImg.style.visibility
```

(If you find the preceding line of code a bit daunting, don't worry! I discuss the `style` object further in the next section, and you can find out more about the document object model in general — including how to navigate it using dots, as shown above — in Chapter 3.)

You're Really Stylin' Now!

Cascading style sheets allow Web developers to separate the way information is presented (bold, underlined, red, green, centered, and so on) from the information itself. And because the presentation properties you define using cascading style sheets become a part of the browsers' document object models, you can manipulate those properties dynamically using JavaScript. Listing 12-4 shows you how!

Listing 12-4: Manipulating Cascading Style Sheet Properties

```
<SCRIPT LANGUAGE="JavaScript">

function changeColor(newColor) {
// Changing text color by referencing the UPCTR ID selector
    document.all.UPCTR.style.color = newColor
}

</SCRIPT>

//Defining a cascading style sheet
<STYLE TYPE="text/css">

/* ---------------------------------------------------
Define an ID selector called UPCTR that centers
associated text and displays it in all uppercase.
--------------------------------------------------- */

//Defining the UPCTR ID selector
#UPCTR {
    color= red;
    text-align: center;
    text-transform: uppercase;
}
```

(continued)

Listing 12-4 *(continued)*

```
</STYLE>
</HEAD>

<BODY>

<H1>Changing CSS properties dynamically</H1>

<P ID="UPCTR">
By default, this paragraph displays red, centered, and all
          uppercase because it's associated with a specially
          defined ID selector (called UPCTR).<BR>
</P>
Clicking the buttons changes the value for the color property
          associated with the UPCTR ID selecton.
<P>
<INPUT TYPE="button" VALUE="Change text color: green"
onClick="changeColor('green')">
<INPUT TYPE="button" VALUE="Change text color: red"
onClick="changeColor('red')">
```

In this example, the text you display on a Web page changes from green to red based on whether a user clicks the `Change text color: green` or `Change text color: red` button.

Most of the code you see in Listing 12-4 is HTML and cascading style sheet declarations. The JavaScript code you want to take a look at is the `changeColor()` function:

```
document.all.UPCTR.style.color = newColor
```

This line of JavaScript changes the color of the displayed text by assigning a new color (the color selected by the user and transmitted to `changeColor()` by the `onClick` event handler) to the text element's `color` property.